Think Java

How to Think Like a Computer Scientist

Allen B. Downey

5.1.1

The original form of this book is LATEX source code. Compiling this LATEX source has the eect of generating a device-independent representation of the book, which can be converted to other formats and printed.

The LATEX source for this book is available from: http://thinkapjava.com This book was typeset using LATEX. The illustrations were drawn in xg. All of these are free, open-source programs.

ভূমিকা

“As we enjoy great Advantages from the Inventions of others, we should be glad of an Opportunity to serve others by any Invention of ours, and this we should do freely and generously."

-Benjamin Franklin, quoted in Benjamin Franklin by Edmund S. Morgan.

আমার এই বই লেখার উদ্দেশ্য

১৯৯৯ সাল থেকে যখন আমি কলবি কলেজে পড়াচ্ছিলাম তখন থেকে আমি বইটি লেখা শুরু করেছি, যেটির এখন পঞ্চম সংস্করণ। আমি একটি প্রারম্ভিক কম্পিউটার বিজ্ঞান ক্লাশের চিন্তা করেছিলাম, কিন্তু আমি আমার পছন্দসই কোন পাঠ্যবই পাইনি। সমস্ত বইগুলোই বড়! আমি চাইলেও শিক্ষার্থীদের ৮০০ পৃষ্ঠার এত বড় প্রযুক্তিগত বিষয়বস্তু পড়ার কোন উপায় নেই। এবং আমি এটি চাইও না। উপাদানগুলোর বেশিরভাগই জাভা এবং এর লাইব্রেরি সম্পর্কে অনেক বিস্তারিত লেখা যা শেষ করতে প্রায় এক সেমিষ্টার লেগে যাবে, এবং তাই আমি আমার প্রয়োজন অনুসারে উপাদন খুঁজে পাচ্ছিলাম না।

অন্য একটি সমস্যা আমি খুঁজে পেয়েছি তা হল একটি অবজেক্ট অরিয়েন্টেড প্রোগ্রামের পরিচয় খুবই অপ্রত্যাশিত ছিল। আমরা অবজেক্ট পাওয়ার পর দেখলাম অন্যান্য ছাত্ররা যারা অন্যথায় ভাল করছিল তারা দেয়ালে আঘাত করছে আমরা শুরু, মাঝ অথবা শেষ যেখান থেকেই শুরু করি না কেন।

তাই আমি লিখতে শুরু করলাম। আমি ১৩ দিন ধরে একটি অধ্যায় লিখলাম, এবং ১৪ তম দিনে আমি সম্পাদনা করলাম। এরপর আমি এটি ফটোকপি এবং বাউন্ড করালাম। যখন আমি আমার ছাত্রদের প্রথম দিন ক্লাশে এটি দিলাম, আমি তখন বললাম যে আমি প্রত্যাশা করছি এক সপ্তাহে একটি অধ্যায় পড়তে। অন্য কথায় বলা যায়, তারা এটি পড়তে পারে সাত গুন ধীরে আমি যতসময় ধরে তা লিখেছি তার তুলনায়।

এটির পেছনে যে দর্শন কাজ করছে

এখানে কিছু ধারণা দেয়া হল যার উপর বইটি লেখা হয়েছে:

- শব্দতালিকা হল গুরুত্বপূর্ণ। ছাত্রদের প্রোগ্রামের বিষয়ে কথা বলতে হবে এবং বুঝতে হবে আমি কি বলতে চাচ্ছি। আমি সর্বনিম্ন পরিমান টার্মগুলোর পরিচয় ঘটানোর চেষ্টা করেছি যারা প্রথম ব্যবহার করছে এবং সেগুলোকে প্রতিটি অধ্যায়ের শেষে শব্দকোষে সাজিয়েছি। আমার ক্লাশে আমি কুইজ এবং পরীক্ষায় শব্দতালিকার প্রশ্ন সংযুক্ত করেছি, এবং ছাত্রদের সঠিক টার্মে সংক্ষিপ্ত উত্তর দেয়ার জন্য আদেশ দিয়েছি।

- একটি প্রোগ্রাম লেখার জন্য, ছাত্রদের অ্যালগোরিদম বুঝতে হবে, প্রোগ্রামিং ভাষা বুঝতে হবে, এবং তাদের ডিবাগ করতে সক্ষম হতে হবে। আমার মনে হয় অনেক বই ডিবাগিং উপেক্ষা করে। এই বইয়ের সাথে ডিবাগিং এর জন্য একটি পরিশিষ্ট এবং প্রোগ্রাম ডেভলপমেন্ট এর জন্য একটি পরিশিষ্ট সংযুক্ত করা হয়েছে (যা ডিবাগিং এড়িয়ে যেতে সহায়তা করবে)। আমি ছাত্রদের এই লেখাটি তাড়াতাড়ি পড়ার জন্য এবং প্রয়োজন হলে আবার পড়ার জন্য পরামর্শ দিচ্ছি।
- কিছু ধারণার ভিতরে যেতে কিছু সময় নিতে পারে। এই বইয়ে recursion এর মত আরো কিছু কঠিন ধারণা আছে যা মাঝে মাঝে আসবে। এই ধারণাগুলো নিয়ে আমি চেষ্টা করছি ছাত্রদের রিভিউ এবং

রিইনফোর্স করার জন্য অথবা তারা যদি এটি ধরতে প্রথমবার ব্যর্থ হয় তাহলে একটি সুযোগ দিতে।
- আমি চেষ্টা করছি সর্বনিম্ন পরিমান জাভা ব্যবহার করে কিভাবে সর্বোচ্চ পরিমান প্রোগ্রামিং পাওয়ার পাওয়া যায়। এই বইটির উদ্দেশ্য হল প্রোগ্রমিং শেখা এবং কম্পিউটার বিজ্ঞানের প্রাথমিক কিছু ধারণা শেখা, কিন্তু জাভা নয়। আমি কিছু switch statement এর মত কিছু ল্যাঙ্গুয়েজ ফিচার যেগুলো অপ্রয়োজনীয় সেগুলো বাদ দিয়েছি এবং অনেক লাইব্রেরি যেমন AWT এর মত যা দ্রুত পরিবর্তন এবং প্রতিস্থাপিত হয় সেগুলো এড়িয়ে গেছি।

আমার এই ক্ষুদ্রকরণ উদ্দেশ্যের কিছু সুবিধা আছে। প্রতিটি অধ্যায়ে অনুশীলনী বাদ দিয়ে ১০টি পৃষ্ঠা আছে। আমার ক্লাশে প্রতিটি ছাত্রকে আমি আলোচনা করার আগে এটি পড়তে বলি এবং আমি দেখেছি যে তারা এটি ভালভাবেই করছে এবং তাদের বোধশক্তি ভাল। তাদের প্রস্তুতি ক্লাশের সময়কে অ্যাবসট্রাক্ট মেটেরিয়াল, ক্লাশের অনুশীলনীর, এবং আরো বিষয়ের উপর আলোচনা করার জন্য সুযোগ করে দেয়।

কিন্তু এই ক্ষুদ্রকরণের কিছু অসুবিধাও রয়েছে। এখানে বেশী মজার কিছু নেই। আমার সবগুলো উদাহরণই হল একটি ল্যাঙ্গুয়েজ বৈশিষ্ট্যের প্রাথমিক ধারণা, এবং স্ট্রিং ম্যানিপুলেশন ও গাণিতিক ধারণার সাথে সংশ্লিষ্ট। আমার মনে হয় সেগুলো কিছু মজার জন্য, কিন্তু সেগুলোর অধিকাংশই ছাত্রদের উদ্দীপিত করে গ্রাফিক্স,সাউন্ড এবং নেটওয়ার্ক অ্যাপ্লিকেশনের মত কম্পিউটার বিজ্ঞান সম্পর্কে যা সংক্ষিপ্তাকারে দেয়া আছে।

সমস্যাটি হল অনেক রোমাঞ্চকর বৈশিষ্ট্য হল কম ধারণা বিশিষ্ট অনেক বেশী বিবরণ সমৃদ্ধ। শিক্ষা-সংক্রান্ত, যার অর্থ হল প্রতিদানের চেয়ে অনেক বেশী প্রচেষ্টা। তাই সেখানে উপাদান গুলোর মধ্যে একটি বানিজ্য চলে যেগুলো ছাত্ররা উপভোগ করে এবং যেগুলো অনেক বুদ্ধিমত্তা সমৃদ্ধ সেই উপাদান গুলোর মধ্যে। আমি ক্লাশের জন্য ভালটি খুঁজে পেতে প্রত্যক শিক্ষকের কাছে ব্যাপারটি ছেড়ে দিলাম। সহায্যের জন্য, বইটিতে পরিশিষ্ট সংযুক্ত করা হয়েছে যার মধ্যে গ্রাফিক্স, কিবোর্ড ইনপুট এবং ফাইল ইনপুট সংযুক্ত রয়েছে।

অবজেক্ট-অরিয়েন্টেড   প্রোগ্রামিং

কিছু বই অতি তাড়াতাড়ি অবজেক্টকে পরিচিত করিয়ে দেয়; অন্যগুলো পদ্ধতিগতভাবে ক্রমশ অবজেক্ট অরিয়েন্টেড স্টাইলের দিকে এগিয়ে যায়। এই বই “objects late” অভিগমন ব্যবহার করে।

জাভার অনেক অবজেক্ট-অরিয়েন্টেড বৈশিষ্ট্য পূর্ববর্তী ভাষাগুলোর সমস্যার দ্বারা চালিত হয়েছিল, এবং তাদের প্রয়োগগুলো ইতিহাস দ্বারা প্রভাবিত হয়েছিল। এই বৈশিষ্ট্যগুলোর কিছু ব্যাখ্যা করা খুবই কঠিন যদি ছাত্ররা যে সমস্যা গুলো তারা সমাধান করেছে সেগুলোর সাথে সামঞ্জস্য না থাকে।

অবজেক্ট-অরিয়েন্টেড প্রোগ্রামিং স্থগিত করা আমার উদ্দেশ্য নয়। এর বিপরীতে আমি চেষ্টা করেছি পরবর্তীতে যাওয়ার পূর্ব সংক্ষিপ্তাকারে যত দ্রুত সম্ভব একবারে ছাত্ররা যাতে অনুশীলনের মাধ্যমে অন্তরণ করতে পারে। কিন্তু আমাকে সেখানে পৌছানোর জন্য কিছু সময় দিতে হবে।

কম্পিটার বিজ্ঞানের এপি পরীক্ষা

সাধারণভাবেই যখন কলেজ বোর্ড ঘোষণা করল যে এপি পরীক্ষা জাভাতে পরিবর্তীত হবে, আমি তখন এই বইটি জাভা সংস্করণে উন্নীত করার পরিকল্পনা করলাম। প্রস্তাবায়িত এপি সিলেবাসে আমি দেখলাম যে সেখানে জাভার উপসেট গুলো সবই ছিল, কিন্তু আমি যে উপসেটগুলো পছন্দ করেছিলাম সেগুলোর থেকে অভিন্ন।

জানুয়ারি ২০০৩ এর সময়, আমি বইটির চতুর্থ সংস্করণে এই পরিবর্তনগুলো করেছিলাম:

- আমি এপি সিলেবাসের কভারেজ উন্নয়ন করার জন্য সেকশন সংযুক্ত করেছিলাম।
- ডিবাগিং এবং প্রোগ্রাম উন্নয়নের জন্য আমি পরিশিষ্টের উন্নয়ন করেছিলাম।
- আমি সংগ্রহ করেছিলাম অনুশীলনি, কুইজ, এবং পরীক্ষার প্রশ্নসমূহ যা আমি আমার ক্লাশে ব্যবহার করেছিলাম এবং উপযুক্ত অধ্যায়ের শেষে সেগুলো রেখেছি। আমি আরো কিছু সমস্যা তৈরি করেছি যা এপি পরীক্ষার জন্য প্রস্তুতি নিতে সহায়তা করবে।

অবশেষে ২০১১ সালের আগষ্ট মাসে আমি পঞ্চম সংস্করণ লিখলাম, এবং এর সাথে এপি পরীক্ষার অংশ GridWorld Case Study এর কভারেজ সংযুক্ত করলাম।

বিনামূল্যের বই!

শুরু থেকেই, এই বইটি ব্যবহারকারীদের অনুলিপি, বিতরণ এবং পরিবর্তনের অনুমোদনের লাইসেন্সের অধীনে রয়েছে। পাঠকেরা বইটি বিভিন্ন ফরমেটে ডাউনলোড করতে পারবে এবং স্ক্রিনে এটি পড়তে পারবে ও এটি মুদ্রণ করতে পারবে। শিক্ষকদের যত কপি দরকার তত কপি তারা মুদ্রণ করতে পারবে। এবং যেকেউ বইটি তাদের প্রয়োজন অনুসারে পরিবর্তন করে নিতে পারবে।

জনগন এই বইটি অন্য কম্পিউটার ভাষায় অনুবাদ করতে পারবে (পাইথন এবং আইফেল সহ), এবং অন্যান্য প্রাকৃতিক ভাষায় (স্পেনিশ, ফ্রেঞ্চ এবং জার্মান সহ)। এই সমস্ত উপাদানসমূহের অনেকগুলো মুক্ত লাইসেন্স এর অধীনে সহজলভ্য।

উন্মুক্ত সফটওয়্যার দ্বারা অনুপ্রাণিত হয়ে, আমি বইটি দ্রুত প্রকাশ করার ইচ্ছা পোষণ করেছি এবং মাঝেমাঝে এটি উন্নীতকরণ করেছি। আমি সর্বোচ্চ চেষ্টা করেছি ত্রুটির সংখ্যা কমিয়ে আনতে, কিন্তু এজন্য আমি পাঠকদেরও সাহায্য চাইছি।

সাড়া পড়েছিল অসাধারণ। আমি খবর পেয়েছি যে প্রতিদিন যারা এই বই পড়েছিল এবং অনেক পছন্দ করেছিল তারা “বাগ রিপোর্ট ” পাঠানোর সমস্যাটি নিয়েছিল। মাঝেমাঝে আমি সমস্যার সমাধান করতে পারতাম এবং উন্নত সংস্করণ আপডেট করতাম কয়েক মিনিটের মধ্যেই। আমি বইটির উন্নতকরণ একটি কাজ হিসাবে নিয়েছিলাম, যখনই একবার পুন: পরীক্ষার সুযোগ পেতাম অথবা পাঠকেরা তাদের মতামত দিত তখনই বইটির উন্নয়ন করতাম।

অহো শিরোনাম

আমার অনেক দুর্দশা পোহাতে হয়েছিল এই বইটির শিরোনামের জন্য। আসলে অনেকেই বুঝতে পারে না যে এটি ছিল অনেকাংশে কৌতুক। এই বইটি পড়ে তোমাকে একজন কম্পিউটার বিজ্ঞানীর মত চিন্তা করতে শিখাবে না। এটি সময়, অভিজ্ঞতা, এবং সম্ভবত কিছু বেশী ক্লাশ নেবে।

কিন্তু শিরোনামে সত্য একটি মর্মস্থল আছে: এই বইটি জাভা সম্পর্কিত নয়, এবং এটি শুধুমাত্র প্রোগ্রামিং সম্পর্কিত একটি অংশ। যদি এটি সফল হয় এই বইটি তাহলে চিন্তার একটি উপায় হবে। কম্পিউটার বিজ্ঞানীদের সমস্যা সমাধানের একটি ধারা আছে, এবং নৈপুণ্যতার সাথে সমাধানের একটি উপায় আছে, সেটি হল অদ্বিতীয়, বহুমুখী, এবং শক্তিশালী। আমি আশা করি এই বইটি তার উদ্দেশ্য সম্পর্কে তোমাকে কিছু ধারণা দিবে, এবং কিছু পয়েন্টে তুমি তোমাকে একজন কম্পিউটার বিজ্ঞানী হিসাবে খুঁজে পাবে।
Allen Downey
Needham Massachusetts
July 13, 2011

অবদানকারীর তালিকা

যখন আমি বিনামূল্যের বই লেখা শুরু করলাম, এটি আমাকে অবদানকারীর তালিকা রাখার কথা আমার মনে উদিত করেনি। যখন Jeff Elkner এটি প্রস্তাব করেন এটি খুবই সুস্পষ্ট যে আমি ভ্রান্তির দ্বারা বিব্রত অবস্থার মধ্যে পড়ি। এই তালিকাটি চতুর্থ সংস্করণের সাথে শুরু হয়, তখন উপদেশ দানকারী এবং সংশোধনে অংশগ্রহণকারীর সংখ্যা পূর্ববর্তী সংস্করণের থেকে অনেক ছাড়িয়ে যায়।

যদি তোমার কোন বাড়তি মন্তব্য থাকে তাহলে তাদের পঠিয়ে দাও এই ঠিকানায়:

feedback@greenteapress.com

- Ellen Hildreth, Wellesley College এই বইটি ডাটা স্ট্রাকচার শিখানোর জন্য ব্যবহার করে এবং সে আমাকে ভাল সাজেশন সহ অনেক সংশোধন দিয়েছে।
- Tania Passfield, চতুর্থ অধ্যায়ের শব্দকোষে কিছু বাদ থেকে যাওয়া টার্ম নির্দিষ্ট করেছে যা এখন আর টেক্সটে নেই।
- Elizabeth Wiethoff লক্ষ্য করেছেন যে আমার সিরিজের ($-x^2$) এর ব্যাখ্যা ভুল ছিল। সে এই বইয়ের Ruby সংস্করণে কাজ করেছে।
- Matt Crawford একটি সম্পূর্ণ প্যাচ ফাইল সম্পূর্ণ সংশোধন করে পাঠিয়েছিলো।
- Chi-Yu Li একটি কোড উদাহরণের একটি টাইপো এবং একটি ত্রুটি খুঁজে বার করেছিলো।
- Doan Thanh Nam অধ্যায় ৩ এর একটি উদাহরণ সংশোধন করেছিলো।
- Stijn Debrouwere একটি ম্যাথ টাইপো খুঁজে পেয়েছিলো।
- Muhammad Saied আরব ভাষায় বইটি অনুবাদ করেছিলো, এবং কিছু ত্রুটি খুঁজে বার করেছিলো।
- Marius Margowski একটি কোড উদাহরণে একটি অসঙ্গতি খুঁজে পেয়েছিলো।
- Guy Driesen কিছু টাইপস খুঁজে পেয়েছিল।
- Leslie Klein বর্তমান সময়ে $\exp(-x^2)$ এর সিরিজ ব্যাখ্যায় অন্য একটি ত্রুটি খুঁজে বার করেছে।

অবশেষে, আমি Chris Mayfield এর নিকট কৃতজ্ঞতা স্বীকার করছি বইটির সর্বশেষ সংস্করণে সহযোগিতা করার জন্য। তার যত্নশীল পর্যালোচনা শতশত সংশোধন এবং উন্নতিসাধনের পথপ্রদর্শন করেছে। হাইপারটেক্সট লিংক এবং ক্রস রেফারেন্স, সমস্ত অনুশীলনীর সুসংগত বিন্যাস, এবং কোড উদাহরণে জাভা সিনট্যাক্স সবার দৃষ্টিগোচর করার মত কিছু নতুন বৈশিষ্ট্য সংযুক্ত করা হয়েছে।

সূচীপত্র

অধ্যায় ১

প্রোগ্রাম শেখার পদ্ধতি

তুমি একজন কম্পিউটার বিজ্ঞানীর মত চিন্তা ভাবনা করবে এটা শেখানোই এই বইয়ের মূল লক্ষ্য। কম্পিউটার বিজ্ঞানীদের চিন্তা করার পদ্ধতি আমার পছন্দ কারণ তারা গণিত, প্রকৌশল এবং প্রকৃতি বিজ্ঞানের শ্রেষ্ঠ বৈশিষ্ট্যগুলো কে একত্র করে নেন। যেমন গণিতবিদগণ, কম্পিউটার বিজ্ঞানীগণ তাদের ধারণা তুলে ধরতে প্রথানুযায়ী ভাষা ব্যবহার করে থাকেন (নির্দিষ্ট করে বলতে গেলে গাণিতিক ধারণাসমূহ)। প্রকৌশলীগণ যেমন কোন কিছুর নকশা করে, বিভিন্ন অংশ জোড়া দিয়ে একটি সিস্টেম তৈরি করেন এবং বিকল্প সিস্টেমের সাথে সুবিধা অসুবিধা বিচার বিবেচনা করে থাকেন। বিজ্ঞানীগণ যেমন জটিল কোন সিস্টেম গভীরভাবে নিরীক্ষণ করেন, একটি হাইপোথিসিস দাঁড়া করান, এবং পূর্বাভাসসমূহ পরীক্ষা করেন।

একজন কম্পিউটার বিজ্ঞানীর সবচেয়ে গুরুত্বপূর্ণ দক্ষতা হচ্ছে **সমস্যা-সমাধান** করতে পারা। এ কথার দ্বারা আমি বোঝাতে চাচ্ছি সমস্যা প্রণয়ন করতে পারার ক্ষমতা, তার সমাধান করার সৃষ্টিশীল চিন্তা করা, এবং স্পষ্টভাবে ও নির্ভুলভাবে এর সমাধান দেয়া। এটা প্রতীয়মান হচ্ছে যে, প্রোগ্রাম শেখার পদ্ধতি সমস্যা-সমাধান করার যে দক্ষতা তা অনুশীলন করার একটি চমৎকার সুযোগ। এ কারণেই এই অধ্যায়ের নাম দেয়া হয়েছে "প্রোগ্রাম শেখার পদ্ধতি।"

একদিকে তুমি প্রোগ্রাম করতে শিখবে, যা একটি প্রয়োজনীয় দক্ষতা। অন্যদিকে তুমি প্রোগ্রামিং ব্যবহার করে কোন কিছু অর্জন করতে সমর্থ হবে। আমরা যত সামনের দিকে যাব, বিষয়টি তত বেশী পরিষ্কার হবে।

১.১ প্রোগ্রামিং ভাষা কি?

তুমি যে প্রোগ্রামিং ভাষাটি শিখবে তার নাম হল জাভা, যা তুলনামূলক ভাবে নতুন (১৯৯৫ সালে Sun এটির প্রথম সংস্করণ প্রকাশ করে)। জাভা একটি উচ্চ-স্তরের ভাষার উদাহরণ; অন্যান্য উচ্চ-স্তরের ভাষা যাদের নাম তোমরা হয়ত শুনে থাকবে যেমন: পাইথন, সি অথবা সি++, এবং পার্ল।

তোমরা হয়ত "উচ্চ-স্তরের ভাষা"নাম থেকেই বুঝে গিয়েছ এটি কি, নিম্ন-স্তরের ভাষাও কিন্তু রয়েছে, মাঝে মধ্যে এদেরকে বলা হয় যান্ত্রিক ভাষা অথবা অ্যাসেম্বলি ভাষা। স্বাধীনভাবে বলতে গেলে, কম্পিউটার শুধুমাত্র নিম্ন-স্তরের ভাষায় লিখিত প্রোগ্রাম দ্বারা চলতে পারে। যে কারণে, উচ্চ-স্তরের ভাষায় লিখিত প্রোগ্রাম চালনা করার আগে নিম্ন-স্তরের ভাষাতে অনুবাদ করে নিতে হয়। এই অনুবাদে সময় নষ্ট হয়, যা উচ্চ-স্তরের ভাষার ছোট একটি অসুবিধা।

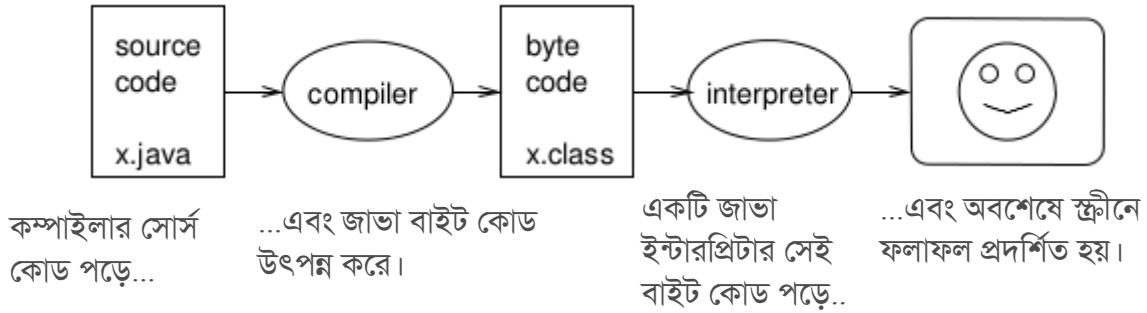
সুবিধা রয়েছে প্রচুর। প্রথমত, উচ্চ-স্তরের ভাষায় প্রোগ্রাম করা অধিকতর সহজ: প্রোগ্রাম লিখতে সময় কম লাগে, পড়তে সহজ এবং সংক্ষিপ্ত, এবং নির্ভুল হয়। দ্বিতীয়ত, উচ্চ-স্তরের ভাষা বহনযোগ্য, মানে হচ্ছে এটি ভিন্ন ভিন্ন কম্পিউটারে চালনা করা যায় এবং এর জন্য অল্পকিছু অথবা কোন প্রকারের পরিবর্তন করতে হয় না। নিম্ন-স্তরের ভাষা শুধুমাত্র এক ধরনের কম্পিউটারে চালনা করা যায়, এবং একটি প্রোগ্রাম অন্য কম্পিউটারে চালাতে হলে নতুন করে লিখতে হয়।

এই সুবিধার কারণে প্রায় সকল প্রোগ্রাম উচ্চ-স্তরের ভাষায় লিখা হয়ে থাকে। নিম্ন-স্তরের ভাষা শুধুমাত্র কিছু বিশেষ অ্যাপ্লিকেশনে ব্যবহার করা হয়ে থাকে।

একটি প্রোগ্রামকে দুইভাবে অনুবাদ করা যায়; ইন্টারপ্রিটিং এবং কম্পাইলিং। ইন্টারপ্রিটার হচ্ছে এমন একটি প্রোগ্রাম

যা উচ্চ-স্তরের ভাষা পড়ে এবং সে অনুযায়ী কাজ করে। কার্যত এটি এক লাইন করে প্রোগ্রাম অনুবাদ করে, এবং বিপরীতভাবে লাইন পড়ে ও কমান্ড অনুযায়ী কাজ করে।

কম্পাইলার হচ্ছে এমন একটি প্রোগ্রাম যা উচ্চ-স্তরের ভাষা পড়ে এবং পুরোটা একবারে অনুবাদ করে, এবং কাজটি হয় কোন প্রকার কমান্ড পরিচালনা করার পূর্বে। প্রায়শই তোমরা আলাদা ধাপ হিসেবে প্রোগ্রাম কম্পাইল করে থাকবে এবং পরবর্তীতে কম্পাইলকৃত কোড চালনা করে থাকবে। এক্ষেত্রে উচ্চ-স্তরের ভাষাকে বলা হয়ে থাকে সোর্স কোড, এবং অনুবাদকৃত প্রোগ্রামকে বলা হয় অবজেক্ট কোড অথবা এক্সিকিউটেবল।

জাভা কম্পাইল এবং ইন্টারপ্রিট দুটোই করা হয়। যান্ত্রিক ভাষায় প্রোগ্রাম অনুবাদ করা ব্যতীত, জাভা কম্পাইলার বাইট কোড উৎপন্ন করে। বাইট কোড যান্ত্রিক ভাষার মত ইন্টারপ্রিট করা সহজ (এবং দ্রুত), কিন্তু এটি উচ্চ-স্তরের ভাষার মত বহনযোগ্য। তাই এটি একটি কম্পিউটারে কম্পাইল করে তার বাইট কোড অন্য কম্পিউটারে স্থানান্তর করা যায়, এবং তারপর অন্য কম্পিউটারে বাইট কোড ইন্টারপ্রিট করা যায়। এই সক্ষমতা জাভার একটি সুবিধা যা অন্যান্য উচ্চ-স্তরের ভাষাতে বিদ্যমান নেই।

এই প্রক্রিয়া দেখে অনেক জটিল মনে হলেও, প্রায় সকল প্রোগ্রামিং ডেভেলপমেন্ট পরিবেশে এই ধাপসমূহ তোমার জন্য স্বয়ংক্রিয়ভাবে সম্পন্ন হবে। সাধারণত এটি কম্পাইল করে চালানোর জন্য তোমাকে শুধুমাত্র প্রোগ্রামটি লিখতে হবে এবং একটি বাটন চাপতে হবে অথবা একটি কমান্ড লিখতে হবে। অন্যদিকে, প্রোগ্রামের পেছনে যে সকল কাজ হচ্ছে তা জানা ভাল, যাতে কোন ধরনের সমস্যা হলে তা সহজেই বের করা সম্ভব হয়।

১.২ প্রোগ্রাম কী?

প্রোগ্রাম হচ্ছে পর পর সাজানো কিছু নির্দেশনা যা নির্দিষ্টভাবে বলে দেয় কিভাবে একটি গণনার কাজ করতে হবে। কাজটি হতে পারে গণিত সংক্রান্ত, যেমন কোন সমীকরণের সমাধান করা অথবা কোন এক বহুপদীর মূল বের করা, কিন্তু এটি সাঙ্কেতিক গণনাও হতে পারে, যেমন নথির মধ্যে কোন লেখা খোঁজা অথবা প্রতিস্থাপনা করা অথবা একটি প্রোগ্রাম কম্পাইল করা।

যে নির্দেশনার কথা বলা হচ্ছে তাকে আমরা বলব স্টেটমেন্টস (**statements**), যা ভিন্ন ভিন্ন প্রোগ্রামিং ভাষার জন্য ভিন্ন ভিন্ন রকমের হয়ে থাকে, কিন্তু কিছু মৌলিক অপারেশন রয়েছে যা প্রায় সকল প্রোগ্রামিং ভাষার জন্য একই রকম:

input: (ইনপুট) কীবোর্ড থেকে, অথবা ফাইল থেকে, অথবা অন্য কোন ডিভাইস থেকে ডাটা গ্রহন করা।
output: (আউটপুট) স্ক্রীনে ডাটা প্রদর্শন করা অথবা কোন ফাইলে অথবা অন্য কোন ডিভাইসে ডাটা প্রেরণ করা।
math: (ম্যাথ) মৌলিক গণিত সংক্রান্ত অপারেশন সম্পাদন করা যেমন যোগ করা এবং গুণ করা।
testing: (টেস্টিং) নির্দিষ্ট কিছু শর্ত পরীক্ষা করা এবং ক্রমানুসারে উপযুক্ত স্টেটমেন্ট চালনা করা।

repetition: (রিপিটেশন) সাধারণত কিছু পরিবর্তনের মাধ্যমে, কিছু কার্যপ্রণালী পুন:পুন সম্পাদন করা।

মোটামুটি এগুলোই। তুমি এখন পর্যন্ত যত প্রোগ্রাম ব্যবহার করেছ তা যতই জটিল হোক না কেন এসকল অপারেশন সম্পাদন করতে পারবে এমন স্টেটমেন্ট ব্যবহার করেই তা তৈরি করা হয়েছে। সুতরাং, অন্যভাবে প্রোগ্রামিং এর বর্ণনা দিতে গেলে বলতে হয়, এটি হচ্ছে বড় এবং জটিল কোন কাজকে ভেঙ্গে ছোট করে নিয়ে আসার প্রক্রিয়া, এ প্রক্রিয়া চলতে থাকে ততক্ষণ পর্যন্ত যতক্ষণ না কাজটিকে এসকল মৌলিক অপারেশন দ্বারা সম্পাদন করা যায়।

১.৩ ডিবাগিং কী?

অদ্ভুত কারণে প্রোগ্রামিং এর ক্রুটিসমূহকে বলা হয় **বাগ** এবং এই ক্রুটি খুঁজে বের করা এবং ক্রুটি দূর করার পদ্ধতিকে বলা হয় **ডিবাগিং**।

একটি প্রোগ্রামে তিন ধরনের ক্রুটি ঘটতে পারে এবং এটি দ্রুত পৃথক করে খুঁজে বের করা প্রয়োজন।

১.৩.১ সিনট্যাক্স ক্রুটি

কম্পাইলার শুধুমাত্র সিনট্যাক্স অনুযায়ী নির্ভুল এমন সব প্রোগ্রাম অনুবাদ করতে পারে; অন্যথায় কম্পাইল প্রক্রিয়া ব্যর্থ হবে এবং তুমি তোমার প্রোগ্রামটি চালাতে পারবে না। **Syntax** হচ্ছে তোমার প্রোগ্রামের কাঠামো এবং সেই কাঠামোর নিয়মকানুন।

উদাহরণস্বরূপ, ইংরেজীতে একটি বাক্য শুরু হবে অব্যশই বড় হাতের অক্ষর দিয়ে এবং শেষ হবে একটি বিরতি দিয়ে। this sentence contains syntax error. পূর্বের বাক্যটিতে সিনট্যাক্স ক্রুটি রয়েছে।

বেশীরভাগ পাঠকের জন্য স্বল্প সিনট্যাক্স ক্রুটি তেমন কোন বড় সমস্যা নয়। একারণেই আমরা *Edward Estlin Cummings* এর কবিতা ঝামেলা ছাড়াই পড়তে পারি।

কম্পাইলার কিন্তু বেশ বোকা। যদি তোমার প্রোগ্রামে একটি অতি সামান্য সিনট্যাক্স ক্রুটি থাকে, তাহলে কম্পাইলার ক্রুটি বার্তা দেখাবে এবং বন্ধ হয়ে যাবে, এবং তুমি তোমার প্রোগ্রামটি চালাতে পারবে না।

আরও খারাপ ব্যাপার হচ্ছে ইংরেজীতে সিনট্যাক্সের জন্য যত নিয়মকানুন আছে তার থেকে বেশী রয়েছে জাভাতে, এবং কম্পাইলার থেকে যে ক্রুটি বার্তা তুমি পাবে তা খুব বেশী সহায়ক নয়। প্রোগ্রামিং শুরু করার পর প্রথম দিকে, তুমি সম্ভবত সিনট্যাক্স ক্রুটি খুঁজে বের করার পেছনে অধিক সময় ব্যয় করবে। ধীরে ধীরে তোমার অভিজ্ঞতা বাড়বে এবং একসময় তুমি ভুল কম করবে এবং দ্রুততার সাথে ক্রুটি খুঁজে বের করতে পারবে।

১.৩.২ রান-টাইম ক্রুটি

দ্বিতীয় ধরনের ক্রুটি হল রান-টাইম ক্রুটি, এই নাম দেয়ার কারণ হচ্ছে এ ধরনের ক্রুটি তুমি কেবলমাত্র প্রোগ্রাম চালু করার পর ঘটবে। জাভাতে, রান-টাইম ক্রুটি ঘটে যখন অনুবাদক বা ইন্টারপ্রিটার বাইট কোড চালনা করে এবং সেই মুহূর্তে কোন গণ্ডগোল ঘটে।

জাভা একটি নিরাপদ ভাষা, এর মানে হচ্ছে কম্পাইলার অনেক ক্রুটি নিজ থেকেই বুঝতে পারে। তাই জাভাতে রান-টাইম ক্রুটি বিরল, বিশেষ করে ছোট প্রোগ্রামসমূহের জন্য।

জাভাতে, রান-টাইম ক্রুটিকে বলা হয় এক্সেপশন (exceptions), এবং প্রায় সকল ডেভেলপমেন্ট পরিবেশেই এগুলো একটি উইন্ডো অথবা ডায়ালগ বক্স আকারে প্রদর্শিত হয়, যাতে বলা থাকে কি ঘটেছে এবং ঘটার সময় অথবা ঘটার পরে প্রোগ্রামটি কি করছিল। এই তথ্যসমূহ ডিবাগিং করার জন্য সহায়ক।

১.৩.৩ যুক্তিগত এবং শব্দর্থগত ক্রুটি

তৃতীয় ধরনের ক্রুটি হল যুক্তিগত অথবা শব্দার্থগত (Logic or Semantic)। যদি তোমার প্রোগ্রামে যুক্তিগত ক্রুটি ঘটে, তাহলে এটি কম্পাইল হয়ে রান হবে কোন ধরনের ক্রুটি বার্তা প্রদর্শন ছাড়াই, কিন্তু এটি সঠিকভাবে কাজ করবে না। সহজ কথায়, এটি সেই কাজই করবে যা তুমি তাকে করতে বলবে।

সমস্যাটি হল তুমি যেই প্রোগ্রামটি লিখেছ সেটি আসলে তুমি লিখতে চাও নি। Semantics, অথবা প্রোগ্রামটির মর্মার্থ ভুল ছিল। যুক্তিগত ক্রুটি খুঁজে বের করতে কিছুটা কৌশলী হতে হবে কারণ তোমাকে তখন প্রোগ্রামের ফলাফল দেখে বুঝতে হবে প্রোগ্রামটি কেন এই ফলাফল দিচ্ছে এবং সে অনুযায়ী প্রোগ্রামের উল্টো দিক থেকে কাজ করতে হবে।

১.৩.৪ পরীক্ষামূলক ডিবাগিং

এই শ্রেণিতে তোমরা সবচেয়ে গুরুত্বপূর্ণ যে জিনিসটি অর্জন করতে পারবে তা হল ডিবাগিং। ডিবাগিং বিষয়টি হতাশাব্যাঞ্জক হওয়া স্বত্বেও, এটি প্রোগ্রামিং এর সবচেয়ে মজার, প্রতিদ্বন্দ্বিতাপূর্ণ, এবং মূল্যবান একটি অংশ।

ডিবাগিং অনেকটা গোয়েন্দাগিরির মত। তুমি অনেক রহস্যের সূত্রের সম্মুখীন হবে এবং তার মাধ্যমে আনুমানিক সিদ্ধান্ত নিয়ে প্রদর্শিত ফলাফলের সমাধান খুঁজে বের করবে।

ডিবাগিং কে অনেকটা গবেষণামূলক বিজ্ঞানের মতও বলা যেতে পারে। যখন তুমি বুঝবে কোথায় গণ্ডগোল, তখন তুমি তোমার প্রোগ্রাম পরিবর্তন করে পুনরায় চেষ্টা করতে পারবে। যদি তোমার অনুমান ঠিক হয়ে থাকে, তাহলে তুমি তোমার প্রোগ্রামের পরিবর্তনের ফলাফল আগে থেকেই বুঝতে সক্ষম হবে, এবং সক্রিয় একটি প্রোগ্রাম তৈরির কাছাকাছি পৌঁছে যাবে। যদি তোমার অনুমান ভুল হয়, তোমাকে নতুন একটি অনুমান করতে হবে। যেমনটা শার্লক হোমস বলেছিল: *"যখন তুমি অসম্ভবকে দূর করতে পারবে, তারপর যা বাকি থাকবে, সেটা অকল্পনীয় হলেও, সত্য।"* (আর্থার কোনান ডয়েলের *দ্যা সাইন অফ ফোর* থেকে।)

কিছু মানুষের জন্য, প্রোগ্রামিং এবং ডিবাগিং একই জিনিস। অর্থাৎ, প্রোগ্রামিং হল এমন একটি প্রক্রিয়া যেখানে একটি প্রোগ্রামকে চাহিদা অনুযায়ী ফলাফল না পাওয়া পর্যন্ত ধীরে ধীরে ডিবাগ করা হতে থাকে। ধারণাটি হল তুমি সর্বদা একটি সক্রিয় প্রোগ্রাম নিয়ে কাজ শুরু করবে যা কিছু না কিছু ফলাফল প্রদর্শন করে, এবং সেটাতে তুমি কিছু ছোট ছোট পরিবর্তন করবে, প্রয়োজন অনুযায়ী ডিবাগ করবে, যাতে তোমার কাছে সবসময় একটি সক্রিয় প্রোগ্রাম থাকে।

উদাহরণস্বরূপ, লিনাক্স হচ্ছে হাজার হাজার লাইনের কোড সম্বলিত একটি অপারেটিং সিস্টেম, কিন্তু এটি সাধারণ প্রোগ্রাম হিসেবে তৈরি হয়েছিল যা লিনুস তোরভাল্ডস ইন্টেল 80386 চিপ এক্সপ্লোর করার জন্য ব্যবহার করতেন। ল্যারি গ্রীনফিল্ড এর মতে, "লিনুস এর অনেকগুলো প্রকল্পের মধ্যে একটি ছিল এমন যা ফলাফল হিসেবে AAAA এবং BBBB প্রদর্শন করত। পরবর্তীতে এটিই লিনাক্স হিসেবে প্রসূত হয়।" (*The Linux Users' Guide* Beta Version 1 হতে)।

পরবর্তী অধ্যায়ে আমি ডিবাগিং এবং অন্যান্য প্রোগ্রামিং অনুশীলন সম্পর্কে আরও সাজেশন দিব।

১.৪ আনুষ্ঠানিক এবং প্রাকৃতিক ভাষা

প্রাকৃতিক ভাষা হচ্ছে যার দ্বারা মানুষ কথা বলে যেমন ইংরেজী, স্প্যানিশ, এবং ফ্রেঞ্চ। এগুলো কোন মানুষ ডিজাইন করে নি (যদিও মানুষ এগুলোর উপর ক্রম আরোপ করেছে); এটির বিবর্ধন হয়েছে প্রাকৃতিক ভাবে।

আনুষ্ঠানিক ভাষা মানুষের দ্বারা সুনির্দিষ্ট অ্যাপ্লিকেশনের জন্য ডিজাইন করা হয়েছে। যেমন, গণিতবিদগণ যেসকল নোটেশন ব্যবহার করে থাকে তা আনুষ্ঠানিক ভাষা যা সংখ্যা এবং চিহ্নের মাঝে সম্পর্ক বোঝাতে সক্ষম। রসায়নবিদগণ অণুর রাসায়নিক কাঠামো প্রকাশ করার জন্য আনুষ্ঠানিক ভাষা ব্যবহার করে থাকেন। এবং সবচেয়ে গুরুত্বপূর্ণভাবে:

> **প্রোগ্রামিং ভাষা হচ্ছে এমন আনুষ্ঠানিক ভাষা যা ডিজাইন করা হয়েছে স্পষ্টভাবে গাণিতিক হিসাব প্রকাশের জন্য।**

আনুষ্ঠানিক ভাষার সিনট্যাক্স এর ব্যাপারে কঠোর নিয়মকানুন রয়েছে। উদাহরণস্বরূপ, 3+3 = 6 সিনট্যাক্স অনুযায়ী একটি সঠিক গাণিতিক স্টেটমেন্ট, কিন্তু 3$ = তা নয়। এছাড়াও, H_2O সিনট্যাক্স অনুযায়ী একটি সঠিক রাসায়নিক নাম, কিন্তু $_2Zz$ তা নয়।

সিনট্যাক্স এর নিয়ম দু'ভাবে হয়ে থাকে, টোকেন এবং কাঠামো সংক্রান্ত। টোকেন হচ্ছে ভাষার একটি মূল উপাদান, যেমনটা শব্দ এবং সংখ্যা এবং রাসায়নিক উপাদান। 3$ = এর একটি সমস্যা হল, $ গণিতে একটি বৈধ টোকেন নয় (আমি যতটুকু জানি)। একইভাবে $_2Zz$ বৈধ নয় কারণ Zz নামক কোন সংক্ষিপ্ত উপাদান নেই।

দ্বিতীয় ধরনের সিনট্যাক্স নিয়ম এর অধীনে রয়েছে একটি স্টেটমেন্ট এর কাঠামো; অর্থাৎ, টোকেন সাজানোর পদ্ধতি। 3$ = উপাদানটি কাঠামোগত ভাবে অবৈধ, কারণ তুমি একটি সমীকরণের শেষে সমান চিহ্ন দিতে পারবে না। একইভাবে, অণুর নামের ক্ষেত্রে, উপাদানের নামের পর সাবস্ক্রিপ্ট দিতে হয়, পূর্বে নয়।

যখন তুমি ইংরেজীতে একটি বাক্য অথবা আনুষ্ঠানিক ভাষায় একটি স্টেটমেন্ট পড়, তোমাকে সেই বাক্যটির কাঠামো চিন্তা করতে হবে (যদিও প্রাকৃতিক ভাষায় তুমি এই কাজটি অবচেতন মনেই করে থাক)। এই কার্যপ্রণালীকে বলা হয় পার্সিং **(parsing)**.

আনুষ্ঠানিক এবং প্রাকৃতিক ভাষার বৈশিষ্ট্যাবলি একই ধরনের---টোকেন, কাঠামো, সিনট্যাক্স এবং সিমানটিক্স---হওয়া সত্ত্বেও এদের মধ্যে পার্থক্য রয়েছে।

অস্পষ্টতা (ambiguity): প্রাকৃতিক ভাষা খুবই অস্পষ্ট, যা মানুষ প্রাসঙ্গিক সূত্র এবং অন্যান্য তথ্যের মাধ্যমে সুরাহা করে। আনুষ্ঠানিক ভাষা যাতে স্পষ্ট হয় সেভাবেই ডিজাইন করা হয়েছে, যার মানে হল বিষয়বস্তু নির্বিশেষে যে কোন স্টেটমেন্ট এর শুধুমাত্র একটি অর্থ থাকবে।

আতিশয্য (redundancy): সন্দেহ দূর করতে এবং ভুল বুঝাবুঝি কমাতে, প্রাকৃতিক ভাষা প্রায়শই প্রয়োজনাতিরিক্ত হয়ে থাকে। আনুষ্ঠানিক ভাষা অধিকতর সংক্ষিপ্ত হয়।

আক্ষরিকতা (literalness): প্রাকৃতিক ভাষাতে প্রচুর উপভাষা এবং রুপকালঙ্কার ব্যবহার করা হয়। আনুষ্ঠানিক ভাষা আক্ষরিক অর্থ প্রকাশ করে।

যে সকল মানুষ প্রাকৃতিক ভাষা ব্যবহার করে বড় হয়েছে (সকলেই) তারা প্রায়শই আনুষ্ঠানিক ভাষার সাথে সমন্বয় করতে গিয়ে সমস্যায় পরে। কিছু দিক থেকে প্রাকৃতিক এবং আনুষ্ঠানিক ভাষার মধ্যে পার্থক্য হচ্ছে অনেকটা কবিতা এবং গল্পের মধ্যকার পার্থক্যের মত।

কবিতা: শব্দগুলো ব্যবহৃত হয় তার ধ্বনি এবং অর্থের জন্য, এবং পুরো কবিতা একসাথে একটি ভাবোদ্দীপক আবহ তৈরী করে। অস্পষ্টতা খুবই সাধারণ এবং স্বেচ্ছাকৃত।

গল্প: শব্দের আক্ষরিক অর্থ অধিক গুরুত্বপূর্ণ এবং কাঠামো আরো অর্থ প্রকাশ করতে সহায়তা করে।
প্রোগ্রাম: একটি কম্পিউটার প্রোগ্রামের অর্থ সন্দেহমুক্ত এবং আক্ষরিক, এবং সম্পূর্ণ বিশ্লেষণের মাধ্যমে এর টোকেন এবং কাঠামো সম্পর্কে অবগত হওয়া যায়।

প্রোগ্রাম (এবং কিছু আনুষ্ঠানিক ভাষা) পড়ার জন্য এখানে কিছু সাজেশন দেয়া হল। প্রথমত, মনে রাখতে হবে যে আনুষ্ঠানিক ভাষা প্রাকৃতিক ভাষার তুলনায় অনেক ভাবগাম্ভীর্যপূর্ণ, সুতরাং এটা পড়তে সময় বেশী প্রয়োজন। সেই সাথে কাঠামোও গুরুত্বপূর্ণ, তাই উপর থেকে নিচে, বাম থেকে ডানে পড়ে যাওয়া ভাল ধারণা নয়। এর পরিবর্তে, তোমার মাথায় প্রোগ্রাম পার্স করা এবং টোকেন সনাক্ত করা এবং কাঠামো অনুবাদ করা শেখ। সব শেষে, তোমরা মনে রাখবে যে, বিস্তারিত বিবরণ অবশ্যই একটি গুরুত্বপূর্ণ ব্যাপার। ছোট খাট জিনিস যেমন, বানান ত্রুটি এবং অশুদ্ধ যতিচিহ্ন, যা তুমি হয়ত প্রাকৃতিক ভাষায় এড়িয়ে যেতে পারবে, কিন্তু আনুষ্ঠানিক ভাষায় এ ধরনের ভুল বিশাল পার্থক্য সৃষ্টি করে।

১.৫ প্রথম প্রোগ্রাম

ঐতিহ্যগতভাবে মানুষ নতুন ভাষায় প্রথম যে প্রোগ্রামটি লিখে তা হল “hello world” কারণ এটি শুধুমাত্র “hello world” শব্দ দুটি স্ক্রীনে প্রদর্শন করে। জাভাতে এই প্রোগ্রামটি হবে এরকম:

```
class Hello {

        // main: generate some simple output

        public static void main(String[] args) {
                System.out.println("Hello, world.");
        }
}
```

এই প্রোগ্রামে এমন কিছু উপাদান রয়েছে যা অনভিজ্ঞ যে কারও কাছে ব্যাখ্যা করা কষ্টসাধ্য, কিন্তু এটি টপিক এর প্রাকদর্শন প্রদান করে যা আমরা পরবর্তীতে দেখব।

জাভা প্রোগ্রাম তৈরি হয় **class definition** দ্বারা, যার ধরন হচ্ছে:

```
class CLASSNAME {
```

```
        public static void main (String[] args) {
                STATEMENTS
        }
}
```

এখানে CLASSNAME নির্দেশিত করছে একটি নাম যা প্রোগ্রামার নিজে পছন্দ করেছে। এই উদাহরণের class name হচ্ছে Hello.

Main হচ্ছে একটি পদ্ধতি (method), যা একটি সংকলনকৃত স্টেটমেন্টের নাম। main একটি বিশেষ নাম; এটি প্রোগ্রাম নির্বাহের শুরুর অবস্থান চিহ্নিত করে। যখন প্রোগ্রাম চালু হয়, এটি main এর প্রথম স্টেটমেন্ট থেকে শুরু হয় এবং শেষ স্টেটমেন্টে গিয়ে সমাপ্ত হয়।

main এর অসংখ্য স্টেটমেন্ট থাকতে পারে, তবে উদাহরণে ছিল একটি। এটা হল print statement, এর মানে হল এটি স্ক্রীনে একটি বার্তা প্রদর্শন করবে। এটা বিভ্রান্তিকর যে print এর মানে বুঝায় "স্ক্রীনে কিছু প্রদর্শন করা" অথবা "মুদ্রণ যন্ত্রে কোন কিছু মুদ্রণের জন্য পাঠানো"। এই বইয়ে আমি মুদ্রণযন্ত্রে কিছু পাঠানোর জন্য বেশী কিছু বলব না; আমরা আমাদের মুদ্রণের কাজ স্ক্রীনেই প্রদর্শন করব। print স্টেটমেন্ট শেষ হয় সেমিকোলন দ্বারা (;)।

System.out.println হচ্ছে জাভা লাইব্রেরী প্রদত্ত একটি পদ্ধতি। লাইব্রেরী হচ্ছে class এবং method definition এর সমন্বয়।

একত্রিত করার জন্য জাভা ব্যবহার করে থাকে ({ এবং }) বন্ধনী। সর্বাধিক বাইরের বন্ধনী (লাইন ১ এবং ৮) ধারণ করে class definition, এবং অভ্যন্তরীন বন্ধনীতে থাকে main এর definition.

তৃতীয় লাইন শুরু হয়েছে // দ্বারা। এর মানে হচ্ছে এই লাইনটি একটি মন্তব্য, যা তুমি প্রোগ্রামের কোড এর মাঝে কিছু ব্যাখ্যা করার জন্য ব্যবহার করতে পার। যখন কম্পাইলার // দেখবে, তখন এটি লাইন শেষ না হওয়া পর্যন্ত সবকিছু উপেক্ষা করবে।

১.৬ শব্দকোষ

সমস্যা-সমাধান: সমস্যা প্রণয়নের প্রক্রিয়া, সমাধান বের করা, এবং সমাধান প্রকাশ করা।

উচ্চ-স্তরের ভাষা: জাভার মত প্রোগ্রামিং ভাষা যা মানুষের পড়া এবং লেখার জন্য সহজভাবে ডিজাইন করা হয়েছে।

নিম্ন-স্তরের ভাষা: কম্পিউটারের জন্য সহজভাবে ডিজাইন করা প্রোগ্রামিং ভাষা। "এটি যান্ত্রিক ভাষা" অথবা "অ্যাসেম্বলি ভাষা" হিসেবেও পরিচিত।

আনুষ্ঠানিক ভাষা: বিশেষ ভাষা যা মানুষ ব্যবহার করে কোন গাণিতিক ধারণা অথবা কম্পিউটার প্রোগ্রাম উপস্থাপনার জন্য। সকল প্রোগ্রামিং ভাষাই আনুষ্ঠানিক ভাষা (formal language)।

প্রাকৃতিক ভাষা: মানুষের ব্যবহৃত যে কোন ভাষা যা প্রাকৃতিকভাবে প্রসূত।

বহনযোগ্যতা: প্রোগ্রামের এমন একটি বৈশিষ্ট্য যার কারণে এটি একাধিক ধরনের কম্পিউটারে চালিত হতে পারে।

ইন্টারপ্রিট: উচ্চ-স্তরের ভাষায় একটি প্রোগ্রাম চালানোর জন্য প্রতি মুহূর্তে এক লাইনের অনুবাদ।

কম্পাইল: উচ্চ-স্তরের ভাষা থেকে নিম্ন-স্তরের ভাষায় একটি প্রোগ্রাম অনুবাদ করা, যা পরবর্তীতে প্রোগ্রাম চালানোর জন্য সহায়ক হবে।

সোর্স-কোড: উচ্চ-স্তরের ভাষায় একটি প্রোগ্রাম, কম্পাইল করার পূর্বে।

অবজেক্ট কোড: প্রোগ্রাম অনুবাদের পর, কম্পাইলারের আউটপুট।

এক্সিকিউটেবল: অবজেক্ট কোডের অন্য এক নাম যা চালনার জন্য তৈরী।

বাইট কোড: বিশেষ ধরনের অবজেক্ট কোড যা জাভায় ব্যবহৃত হয়। বাইট কোড একটি নিম্ন-স্তরের ভাষার অনুরূপ, কিন্তু এটি উচ্চ-স্তরের ভাষার মত বহনযোগ্য (portable)।

স্টেটমেন্ট: প্রোগ্রামের একটি অংশ যা একটি গাণিতিক কার্যপ্রণালী সুনির্দিষ্ট করে।

প্রিন্ট স্টেটমেন্ট: একটি স্টেটমেন্ট যা স্ক্রীনে আউটপুট প্রদর্শন করতে সহায়তা করে।

মন্তব্য: প্রোগ্রামের একটি অংশ যেখানে প্রোগ্রাম সম্পর্কে তথ্য থাকে, কিন্তু প্রোগ্রাম চালনার সময় ব্যবহৃত হয় না।

পদ্ধতি: স্টেটমেন্টের একটি সংগ্রহ।

লাইব্রেরী: ক্লাস এবং মেথড ডেফিনেশনের একটি সংগ্রহ।

বাগ: প্রোগ্রামের ত্রুটি।
সিনট্যাক্স: প্রোগ্রামের কাঠামো।

সিমানটিক্স: প্রোগ্রামের অর্থ।

পার্স: প্রোগ্রাম এবং অন্বিত কাঠামো পরীক্ষা করা।

সিনট্যাক্স ত্রুটি: প্রোগ্রামের মধ্যে এমন এক ত্রুটি যা পার্স করতে বাঁধা দেয় (এবং এ কারণে কম্পাইল করা অসম্ভব হয়ে পরে)।

এক্সসেপশন: চালনার সময় একটি ত্রুটি যা প্রোগ্রাম চালনা ব্যার্থ করে। এটাকে রান-টাইম ত্রুটি বলা হয়ে থাকে।

যৌক্তিক ত্রুটি: প্রোগ্রামের এমন এক ত্রুটি যা প্রোগ্রামের সংকল্পিত কাজ ব্যতীত ভিন্ন কাজ করিয়ে থাকে।

ডিবাগিং: তিন ধরনের ত্রুটির যে কোনটি খুঁজে বের করা এবং অপসারণ করা।

১.৭ অনুশীলনী:

অনুশীলনী ১.১: কম্পিউটার বিজ্ঞানীদের একটি বিরক্তিকর অভ্যাস রয়েছে, তা হল, কোন প্রচলিত ইংরেজী শব্দ

ব্যবহার করে কোন কিছুর অর্থ প্রকাশ করা যা আসলে ওই শব্দটির প্রচলিত অর্থ নয়। উদাহরণস্বরূপ, ইংরেজীতে স্টেটমেন্ট (statement) এবং কমেন্ট (comment) একই জিনিস, কিন্তু প্রোগ্রামিং এ এরা ভিন্ন।

প্রতি অধ্যায়ের শেষে কম্পিউটার বিজ্ঞানে ব্যবহৃত বিশেষ কিছু শব্দ এবং বাক্যের অর্থ জানার উদ্দেশ্যে শব্দকোষ জুড়ে দেয়া হয়েছে। যখন তুমি কোন পরিচিত শব্দ দেখবে তখন ভেবে নিও না যে তুমি এর অর্থ সম্পর্কে অবগত!

১। কম্পিউটারের ভাষায়, স্টেটমেন্ট এবং কমেন্টের মধ্যে পার্থক্য কি?
২। একটি প্রোগ্রামের বহনযোগ্যতা (portability) বলতে কি বোঝায়?
৩। এক্সিকিউটেবল (executable) বলতে কি বুঝ?

অনুশীলনী ১.২: কোন কিছু করার পূর্বে খুঁজে বের কর কিভাবে তোমার কম্পিউটারে একটি জাভা প্রোগ্রাম কম্পাইল করে চালাতে হয়। কিছু প্রোগ্রামিং এনভায়রনমেন্ট অনুচ্ছেদ ১.৫ এর উদাহরণের মত নমুনা প্রোগ্রাম প্রদান করে থাকে।
১। "Hello, world" প্রোগ্রামটি টাইপ কর, তারপর কম্পাইল করে চালনা কর।
২। দ্বিতীয় বার্তা হিসেবে একটি প্রিন্ট স্টেটমেন্ট সংযুক্ত কর যা "Hello, world" এর পর প্রদর্শিত হবে। যেমন: "How are you?" তারপর কম্পাইল করে পুনরায় চালনা কর।
৩। প্রোগ্রামের যে কোন স্থানে একটি মন্তব্য সংযুক্ত কর, নতুন করে কম্পাইল কর, এবং চালনা কর। নতুন মন্তব্যটি যেন আউটপুটে কোন সমস্যার সৃষ্টি না করে।

এই অনুশীলনসমূহ তুচ্ছ মনে হতে পারে, কিন্তু এটা সকল প্রোগ্রাম যা নিয়ে আমরা কাজ করব তার প্রারম্ভিক পর্যায়। আত্মবিশ্বাসের সাথে ডিবাগ করতে হলে তোমাকে অবশ্যই তোমার প্রোগ্রামিং এনভায়রনমেন্টে আত্মবিশ্বাসী হতে হবে। কিছু এনভায়রনমেন্টে, কোন প্রোগ্রাম এক্সিকিউট হচ্ছে তার হিসাব রাখা কষ্টকর হয়ে পরে, এবং তুমি হয়ত একটা প্রোগ্রাম ডিবাগ করতে গিয়ে ভুলবশত অন্য একটি করা শুরু করবে। প্রিন্ট স্টেটমেন্ট যোগ (এবং পরিবর্তন) করার মাধ্যমে তুমি সহজে নিশ্চিত হতে পারবে, যে প্রোগ্রামটি তুমি দেখছ সেই প্রোগ্রামটিই চলছে।

অনুশীলনী ১.৩: এটা খুবই ভাল হবে যদি তুমি অধিক সংখ্য ত্রুটি সম্পর্কে অবগত থাক, যাতে করে তুমি বুঝতে পার কম্পাইলার কি ধরনের ত্রুটি বার্তা উৎপন্ন করছে। মাঝে মাঝে কম্পাইলার তোমাকে নিখুঁত ভাবে বলে দিবে কোথায় সমস্যা হচ্ছে, তখন তোমার সেটা ঠিক করাই একমাত্র কাজ। কিন্তু মাঝে মাঝে ত্রুটি বার্তা দেখে ঠিক মত সমস্যা সম্পর্কে অবগত হওয়া যায় না। তোমাকে এই অনুভূতি প্রখর করার জন্য চেষ্টা চালাতে হবে যাতে তুমি বুঝতে সক্ষম হও কোন পরিস্থিতিতে কি ধরনের পদক্ষেপ নিতে হবে।

১। যে কোন একটি বহিঃস্থ বন্ধনী অপসারণ কর।
২। যে কোন একটি অভ্যন্তরীন বন্ধনী অপসারণ কর।
৩। main এর পরিবর্তে লিখ mian।
৪। static শব্দটি অপসারণ কর।
৫। public শব্দটি অপসারণ কর।
৬। system শব্দটি অপসারণ কর।
৭। println এর পরিবর্তে লিখ Println।
৮। println এর পরিবর্তে লিখ print। এটা কৌশলপূর্ণ ব্যাপার কারণ এটি একটি যৌক্তিক ত্রুটি, সিনট্যাক্স ত্রুটি নয়। System.out.print স্টেটমেন্ট বৈধ, কিন্তু তুমি যা চাচ্ছ এটার মাধ্যমে তুমি সেটা পেতেও পার আবার নাও পেতে পার।
৯। একটি বন্ধনী অপসারণ কর। একটি অতিরিক্ত বন্ধনী যোগ কর।

অধ্যায় ২

ভ্যারিয়েবল এবং এর ধরন (Variables and types)

২.১ একাধিকবার মুদ্রণ (More printing)

আমরা main-এর ভেতরে যতবার চাই তত বার স্টেটমেন্ট (statement) প্রিন্ট করতে পারি। উদাহরণস্বরূপ বলা যায়, একাধিক লাইন প্রিন্ট করার জন্য আমরা বলতে পারি:

```
class Hello {

	// Generates some simple output.

	public static void main(String[] args) {
		System.out.println("Hello, world.");	// print one line
		System.out.println("How are you?");	// print one line
	}
}
```

উপরের উদাহরণের মত করে আমরা কমেন্টগুলো প্রতিটি লাইনের শেষে লিখতে পারি আবার এক লাইনেও কমেন্ট লিখতে পারি।

উদ্ধৃতি চিহ্নের ভিতরে ব্যাক্যাংশকে বলে স্ট্রিং (strings)। Strings পর পর অর্থাৎ ক্রমানুসারে অবস্থিত অনেকগুলো চিহ্নের সমন্বয়ে গঠিত। Strings অনেক বর্ণের সমন্বয় হতে পারে, অনেকগুলো নম্বর মিলে হতে পারে, বিরাম চিহ্নের সমন্বয়ও হতে পারে আবার বিশেষ চিহ্নের মাধ্যমেও গঠিত হতে পারে।

"print line" এর সংক্ষিপ্ত রূপ হলো "println"। প্রত্যেক লাইনের শেষে এই "println" কমান্ডটি একটি বিশেষ চিহ্ন যোগ করে, একে বলে নতুন লাইন বা "newline"। এর ফলেই কার্সরটি পরর্বতী লাইনে চলে যায়। এই কমান্ডটি বলে দেয় যে, নতুন টেক্সটটি নতুন লাইনে দেখাতে হবে।

বিভিন্ন প্রিন্ট স্টেটমেন্ট-এর আউটপুটকে একই লাইনে দেখানোর জন্য আমরা নিচের মতো করে কমান্ড লিখতে পারি :

```
class Hello {

	// Generates some simple output.

	public static void main(String[] args) {
		System.out.print("Goodbye, ");
		System.out.println("cruel world!");
	}
}
```

এক্ষেত্রে আমরা এক লাইনে আউটপুট দেখতে পারব “Goodbye, cruel world!” । একটু লক্ষ্য করলে আমরা দেখতে পাবো “Goodbye” এর পর উদ্ধৃতি চিহ্নের আগে ফাঁকা স্থান বা space ব্যবহার করা হয়েছে যা আউটপুটে দেখা যাচ্ছে। অর্থাৎ এটি প্রোগ্রামের আউটপুটের ওপর প্রভাব ফেলে। তবে উদ্ধৃতি চিহ্নের পরের space আউটপুটে কোন প্রভাব ফেলে না। উদাহরণস্বরূপ আমরা লিখতে পারি,

```
 class Hello {
public static void main(String[] args) {
System.out.print("Goodbye, ");
System.out.println("cruel world!");
}
}
```

এই প্রোগ্রাম ঠিক আগের মতোই কম্পাইল এবং রান করবে। কোডের নতুন লাইনের আগের ব্রেক (break) এক্ষেত্রে কোন প্রভাব ফেলবে না। এমনকি যদি আমরা নিচের মতো করে লিখি তাও না।

```
class Hello { public static void main(String[] args) {
System.out.print("Goodbye, "); System.out.println
("cruel world!");} }
```

এভাবে লিখলেও প্রোগ্রাম কাজ করবে তবে এভাবে লিখলে কোড পড়া খুব কঠিন হয়ে যাবে। দৃষ্টিনন্দনভাবে এবং সংগঠিতভাবে প্রোগ্রাম লেখার জন্য নতুন লাইন এবং ফাঁকা স্থানের প্রয়োজনীয়তা অসীম। নতুন লাইন এবং ফাঁকা স্থান প্রোগ্রামটি পড়তে এবং প্রোগ্রামের ভুল বের করতে অনেক সহযোগিতা করে।

২.২ ভ্যারিয়েবল (Variables)

প্রোগ্রামিং ল্যাঙ্গুয়েজের একটি অন্যতম শক্তিশালী দিক হলো ভ্যারিয়েবল-কে বিভিন্নভাবে কাজে লাগানো বা manipulate করা। ভ্যারিয়েবল হলো কোন value সংরক্ষণের নাম সম্বলিত অবস্থান। Value হলো এমন একটি জিনিস যা আমরা প্রিন্ট করতে পারি, সংরক্ষণ করতে পারি এবং অপারেট করতে পারি। যে স্ট্রিং আমরা প্রিন্ট করলাম ("Hello, World.", "Goodbye, ", etc.) তাও value।
Value সংরক্ষণের জন্য আমাদের ভ্যারিয়েবল তৈরি করা প্রয়োজন। যদি যে value আমরা সংরক্ষণ করতে চাই তা স্ট্রিং হয় তাহলে আমাদের ডিক্লেয়ার (declare) করতে হয় নতুন ভ্যারিয়েবলটি একটি স্ট্রিং।

```
String karim;
```

এই স্টেটমেন্ট হলো একটি declaration, কারণ এটি আমাদের কাছে declare করে karim নামক ভ্যারিয়েবল নামের ধরন হলো স্ট্রিং। প্রত্যেক ভ্যারিয়েবলের একটি ধরন থাকে যা নির্দেশ করে ভ্যারিয়েবলটি কি ধরনের ভ্যারিয়েবল বা কি ধরনের value এটি সংরক্ষণ করে। উদাহরণ হিসেবে বলা যায়, int type ভ্যারিয়েবল সংরক্ষণ করে ইন্টিজার value আবার String type ভ্যারিয়েবল সংরক্ষণ করে স্ট্রিং value।

কিছু ভ্যারিয়েবল শুরু হয় বড় হাতের অক্ষরে আবার কিছু ভ্যারিয়েবল শুরু হয় ছোট হাতের অক্ষরে। আমরা পরে এই পার্থক্যের তাৎপর্য জানতে পারবো কিন্তু এখন আর আমরা এই বিষয়টি আলোচনা করবো। আমাদের মনে রাখতে হবে Int বা string- এই ধরনের কোন ভ্যারিয়েবল নেই। কম্পাইলার সম্পর্কে আমরা আগেই জেনেছি, কম্পাইলার এধরনের ক্ষেত্রে আমাদের ত্রুটি প্রদর্শন করবে।

ইন্টিজার ভ্যারিয়েবল তৈরি করার জন্য আমাদের লিখতে হবে int karim; যেখানে karim (করিম) হলো আমাদের ব্যবহার করা নাম যা আমরা ভ্যারিয়েবল হিসেবে ব্যবহার করতে চাই। সাধারণতঃ আমাদের ভ্যারিয়েবলের নাম এমনভাবে নির্বাচন করতে হবে যাতে নামই নির্দেশ করবে আমরা এই ভ্যারিয়েবল নিয়ে কি করতে চাই। উদাহরণস্বরূপ বলা যায়, যদি আমরা নিম্নোক্ত রূপে ভ্যারিয়েবল declare করতে দেখি:

```
String firstName;
String lastName;
int hour, minute;
```

এই ভ্যারিয়েবল দেখে আমরা বুঝতে পারি কি ধরনের values এই ভ্যারিয়েবল সংরক্ষণ করবে। এই উদাহরণ থেকে আমরা আরও বুঝতে পারি কীভাবে একই ধরনের অনেকগুলো ভ্যারিয়েবল একসাথে declare করা যায়। ঘন্টা এবং সেকেন্ড এই দুটি ভ্যারিয়েবলকে আমরা উপরের উদাহরণের মাধ্যমে int type হিসেবে একসাথে declare করতে পেরেছি।

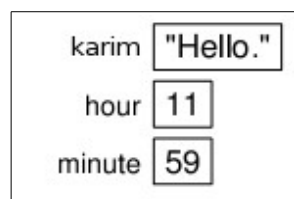
২.৩ অ্যাসাইনমেন্ট (Assignment)

এখন আমরা ভ্যারিয়েবল তৈরি করতে শিখে গেছি, এবার আমাদের দরকার এই value গুলো সংরক্ষণ করা। আমরা এই কাজটি assignment statement-এর সাহায্যে করতে পারি।

```
karim = "Hello.";        // give bob the value "Hello."
hour = 11;               // assign the value 11 to hour
minute = 59;             // set minute to 59
```

এই উদাহরণের মাধ্যমে তিনটি assignment দেখানো হয়েছে আর comments/ মন্তব্যের মাধ্যমে তিনটি ভিন্ন ভিন্ন উপায় বর্ণনা করা হয়েছে, যার মাধ্যমে আমরা assignment statement এর কথা বলতে পারি। হয়তো এইভাবে শব্দ চয়ন করার কারণে অনেক দ্বিধান্বিত হতে পারেন তবে বক্তব্যের মূল ধারণা একই:

- যখন আমরা ভ্যারিয়েবল declare করি, তখন আসলে আমরা একটি সংরক্ষণ স্থান (storage location) -এর নাম তৈরি করি।
- যখন আমরা একটি ভ্যারিয়েবলের assignment তৈরি করি, তখন আমরা আসলে একে একটি value দেই।

কাগজে লিখে যদি আমরা ভ্যারিয়েবলকে বোঝাতে চাই, তাহলে আমাদের ভ্যারিয়েবলের নামটি বাইরে রেখে একটি বক্স তৈরি করতে হবে যার বাইরে থাকবে ভ্যারিয়েবলের নাম এবং মধ্যে থাকবে ভ্যারিয়েবলের value। নিচের ছবিতে আমরা দেখতে পাবো উপরের উদাহরণের তিনটি assignment statement:

সাধারণ নিয়ম অনুসারে, ভ্যারিয়েবলের ধরন আর এর মধ্যে assign করা value এক হতে হবে। আমরা minute-

কে String হিসেবে বা karim-কে integer হিসেবে উল্লেখ করতে পারবো না।

আবার অন্যদিকে, এই নিয়ম অনেক ক্ষেত্রে দ্বিধা তৈরি করে। কারণ আমাদের কাছে অনেক উপায় আছে যার মাধ্যমে আমরা এক ধরনের value-কে অন্য ধরনের value-তে পরিবর্তন (Convert) করতে পারি। আবার কখনো কখনো জাভা (Java) স্বয়ংক্রিয়ভাবেই এই পরিবর্তন করে থাকে। এই সময়ের জন্য আমরা শুধু সাধারণ নিয়মগুলোই মনে রাখার চেষ্টা করবো পরে আমরা ব্যতিক্রম (exceptions) নিয়ে আলোচনা করবো।
দ্বিধাগ্রস্থ হবার আর একটি উৎস হলো, কিছু কিছু strings-কে অনেক সময় integers মনে হয় যদিও তারা তা নয়। যেমন, karim এই string "123" -কে ধারণ করতে পারে, যা কতগুলো characters 1, 2 এবং 3-এর সমন্বয়ে গঠিত কিন্তু এটি number 123 বলতে যা বোঝায় তা নয়।

```
karim = "123";          // legal
karim = 123;            // not legal
```

২.৪ ভ্যারিয়েবল প্রিন্ট করা (Printing variables)

println অথবা print ব্যবহার করে আমরা ভ্যারিয়েবলের value প্রিন্ট করতে পারি:

```
class Hello {
        public static void main(String[] args) {
                String firstLine;
                firstLine = "Hello, again!";
                System.out.println(firstLine);
        }
}
```

এই প্রোগ্রামে "Hello, again!"-এই value-এর জন্য firstLine নামক ভ্যারিয়েবলটি তৈরি করা হয়েছে এবং এরপর value-টি প্রিন্ট করা হয়েছে। যখন আমরা কোন ভ্যারিয়েবল প্রিন্ট করার কথা বলি তার মানে হলো আমরা সেই ভ্যারিয়েবলের value প্রিন্ট করার কথা বলি। ভ্যারিয়েবলের নাম যদি আমরা প্রিন্ট করতে চাই তাহলে আমাদের তা উদ্ধৃতি চিহ্নের ভিতরে রাখতে হবে। যেমন, System.out.println("firstLine");

উদাহরণ হিসেবে বলা যায়, আমরা এভাবেই লিখতে পারি;

```
String firstLine;
firstLine = "Hello, again!";
System.out.print("The value of firstLine is ");
System.out.println(firstLine);
```

এই প্রোগ্রামের আউটপুট হলো;

```
The value of firstLine is Hello, again!
```

এটা আমাদের জন্য আনন্দের খবর যে ভ্যারিয়েবল প্রিন্ট করার জন্য আমাদের যে syntax লিখতে হয় তা ভ্যারিয়েবলের ধরন অনুসারে বিভিন্ন নয়। অর্থাৎ বিভিন্ন ধরনের ভ্যারিয়েবল প্রিন্ট করার জন্য আমাদের একই

syntax লিখতে হবে।

```
int hour, minute;
hour = 11;
minute = 59;
System.out.print("The current time is ");
System.out.print(hour);
System.out.print(":");
System.out.print(minute);
System.out.println(".");
```

এই প্রোগ্রামের আউটপুট হবে;

```
The current time is 11:59.
```

সতর্কতা: একই লাইনে একাধিক value থাকলে, সাধারণত প্রতিটি প্রিন্ট statements-এর জন্য ভিন্ন ভিন্ন println ফাংশন ব্যবহার করা উচিত। কিন্তু আমাদের println ফাংশনের কথা মনে রাখতে হবে। অনেক ক্ষেত্রে দেখা যায়, প্রিন্টের আউটপুটগুলো সঠিকভাবে প্রদর্শন না করে শুধুমাত্র একবার প্রদর্শন করে, যতক্ষণ পর্যন্ত আমরা println ফাংশনকে ব্যবহার না করি। যদি আমরা println ফাংশনটি মুছে ফেলি তাহলে দেখা যাবে, প্রোগ্রামটি সংরক্ষিত আউটপুট প্রদর্শন না করেই বন্ধ হয়ে যাবে।

২.৫ কিওয়ার্ড (Keywords)

কিছুক্ষণ আগেই আমরা বলেছিলাম, আমরা যেকোন নামেই আমাদের ভ্যারিয়েবলের নাম দিতে পারি, কিন্তু এটি পুরোপুরি সত্য নয়। Java-তে কিছু সুনির্দিষ্ট শব্দ সংরক্ষণ করে রাখা হয়েছে কারণ সে শব্দগুলো কম্পাইলার ব্যবহার করে বিভিন্ন প্রোগ্রামের গঠন বিশ্লেষণের জন্য। আমরা যদি সে শব্দগুলো ভ্যারিয়েবল হিসেবে ব্যবহার করি তাহলে কম্পাইলার দ্বিধান্বিত হয়ে পড়বে। এই শব্দগুলোকেই বলে keywords। যার মধ্যে রয়েছে public, class, void, int এবং এরকম আরও অনেক শব্দ।

এই keywords-এর পূর্ণ তালিকা আমরা পেতে পারি http://download.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html -এই লিংক থেকে। ওরাকল Java ডুকমেন্টশনসহ, keywords-এর এই পূর্ণ তালিকা প্রণয়ন করেছে। এই বই-এর পাশাপাশি এই ডুকমেন্টেশন পড়তেও পরার্মশ দেওয়া হলো।

এই লিস্ট মুখস্থ করার চেয়ে সুবিধাজনক হলো বিভিন্ন Java development environments ব্যবহার করা। এতে কোড হাইলাইট করা সহ নানরকম সুবিধা রয়েছে। Java development environment -এ যখন আমরা লিখবো তখন আমাদের লিখিত কোডের বিভিন্ন অংশ বিভিন্ন রং-এর দেখা যাবে। যেমন; কিওয়ার্ড হয়তো দেখাবে নীল, স্ট্রিং হয়তো দেখাবে লাল এবং অন্যান্য কোড হয়তো দেখাবে কালো। আবার যদি আমরা ভ্যারিয়েবলের নাম লিখি তাহলে হয়তো তা সাথে নীল হয়ে যাবে। সম্ভব হলে চলো এখনই তা পরীক্ষা করে দেখি তাহলে আমরা কম্পাইলারের অদ্ভুত আচরণও দেখতে পাবো।

২.৬ অপারেটর (Operators)

অপারেটরস হলো কিছু চিহ্ন যা গণনার কাজে ব্যবহৃত হয় যেমন; যোগ, বিয়োগ বা গুণ চিহ্ন। Java-এর বেশিরভাগ অপারেটরস যা তাদের করার কথা তাই করে কারণ তারা মূলত গাণিতিক চিহ্ন ছাড়া আর কিছু নয়। উদাহরণস্বরূপ , যোগ করার অপারেটরস হলো "+" চিহ্ন, বিয়োগের অপারেটরস হলো "-" চিহ্ন, গুণের হলো "*" চিহ্ন এবং ভাগের হলো "/" চিহ্ন।

```
1+1        hour-1        hour*60 + minute        minute/60
```

এই এক্সপ্রেশন-এ আমরা ভ্যারিয়েবলের নাম এবং নম্বর দুটিই পাচ্ছি। গণনার আগেই ভ্যারিয়েবলগুলো তাদের value দ্বারা প্রতিস্থাপিত হয়।
যোগ, বিয়োগ এবং গুণের ক্ষেত্রে আমরা যেরকম আশা করি সেরকমই ফল পাওয়া যায়। কিন্তু ভাগের ক্ষেত্রে আমরা ভিন্ন চিত্র দেখি:

```
int hour, minute;
hour = 11;
minute = 59;
System.out.print("Number of minutes since midnight: ");
System.out.println(hour*60 + minute);
System.out.print("Fraction of the hour that has passed: ");
System.out.println(minute/60);
```

এই প্রোগ্রামটি নিম্নের আউটপুট প্রদর্শন করবে:

```
Number of minutes since midnight: 719
Fraction of the hour that has passed: 0
```

এক্ষেত্রে প্রথম লাইনটি আমাদের প্রত্যাশা অনুসারে হয়েছে কিন্তু দ্বিতীয় লাইনটি অস্বাভাবিক। এখানে মিনিটের মান 59 এবং 59 -কে ভাগ করা হয়েছে ৬০ দিয়ে যার ফল হলো 0.98333, 0 নয়। এখানে এরকম দেখানোর কারণ হলো Java এখানে ইন্টিজার (integer) নিয়ে কাজ করছে।

আমরা এখন অপারেন্ড (operands)-এর কথা জানবো। Operand-কে নিয়ে কাজ করে operators আবার অন্যভাবে বলা যায় operators কাজ করে operand-এর ওপর। যখন আমাদের দুটি operator-ই হবে integer তখন রেজাল্টও হবে integer। নিয়মানুসারে integer-দ্বয়ের ভাগফল সবসময় পূর্ণ সংখ্যা হবে, যদি কখনো ভাগফল দশমিক সংখ্যা হয় এবং সে দশমিক সংখ্যা পরবর্তী পূর্ণ সংখ্যার খুব কাছাকাছি হয় তাহলেও প্রোগ্রামিং-এর ক্ষেত্রে আমরা ঐ দশমিক সংখ্যার পূর্ণ সংখ্যার অংশটুকু ফল হিসেবে দেখতে পাবো।

এটি আসলে ভগ্নাংশ গণনার চেয়ে শতকার হার বা শতাংশ গণনার বিকল্প পদ্ধতি হতে পারে:

```
System.out.print("Percentage of the hour that has passed: ");
System.out.println(minute*100/60);
```

এর ফলাফল হবে,

```
Percentage of the hour that has passed: 98
```

পূর্বের মতোই আমরা দেখলাম রেজাল্ট পূর্ণ সংখ্যা হয়েছে এবং এখন আমরা বলতে রেজাল্ট আংশিকভাবে সঠিক। আরও সঠিক উত্তরের জন্য আমাদের ভিন্ন ধরনের ভ্যারিয়েবল ব্যবহার করতে হবে। এই ভ্যারিয়েবলকে বলে floatin-point। এটি ভগ্নাংশের value (fractional values) সংরক্ষণ করে। আমরা পরবর্তী অধ্যায়ে এর সম্পর্কে জানবো।

২.৭ অপারেশনের ক্রম (Order of operations)

যখন expression-এ একাধিক operator থাকে, তখন তা মূল্যায়ন করা হয় **rules of precedence**-এর ভিত্তিতে। সম্পূর্ণ explanation of precedence তোমাদের জন্য একটু জটিল হতে পারে, তাই আমরা আমরা এখন খুব সংক্ষেপে শুরু করবো:

- গুণের এবং ভাগের কাজ যোগ এবং বিয়োগের কাজের আগে হবে। অর্থাৎ (2*3-1) -এর ফল হবে 5, 4 নয়। আবার (2/3-1)-এর ফল হবে -1, 1 নয়। আমাদের মনে রাখতে হবে integer division- এর ক্ষেত্রে (2/3)-এর ফল হবে 0।
- যদি সবগুলো অপারেটরস-এর precedence সমান হয় তাহলে তাদের মূল্যায়ন করা হবে বাম থেকে ডানে। অর্থাৎ (minute*100/60) expression-এর জন্য আগের গুনের কাজ হবে, তার ফলে আমরা পাবো (5900/60) এবং এর পরে হবে ভাগের কাজ, ফলে সার্বিক রেজাল্ট হবে 98। যদি অপারেশনের কাজ ডান থেকে বামে করা হয় তাহলে ফল হবে (59*1) অর্থাৎ 59। যা ভুল উত্তর।
- যখন আমরা rules of precedence-কে অগ্রাহ্য করতে চাইবো অথবা আমরা নিশ্চিত হতে পারবো না কোন অপারেটরে কাজ আগে হবে, তখন আমরা parentheses বা বন্ধনী ব্যবহার করতে পারি। Expressions-এ বন্ধনীর ভেতরের অপারেটর-এর কাজ আগে করা হয়। অর্থাৎ 2 *(3-1)-এর ফল হবে 4। এছাড়াও বন্ধনীর ব্যবহারের ফলে expression সহজে পড়ে বোঝা যায়। যেমন; পূর্বের উদাহরণের expression-কে আমরা এভাবে (minute * 100) / 60 বন্ধনীর মাধ্যমেও লিখতে পারি, এরজন্য রেজাল্টের কোন পরিবর্তন হবে না।

২.৮ স্ট্রিং-এর জন্য অপারেটর (Operators for Strings)

সাধারণত আমরা স্ট্রিং-কে নিয়ে কোন গাণিতিক অপারেশন করতে পারি না। এমনকি যদিও স্ট্রিং-গুলো দেখতে নম্বরের মতো হয়। নিচের উদাহরণগুলো লক্ষ্য করি;

```
karim - 1          "Hello"/123          karim * "Hello"
```

এই উদাহরণগুলো সঠিক নয়, যেহেতু আমরা জানি karim হলো স্ট্রিং। এখন আমরা একটু চিন্তা করে বলি তো, উপরের উদাহরণের karim কি integer অথবা string? আসলে এভাবে ভ্যারিয়েবলের ধরন বলা যায় না, ভ্যারিয়বলের ধরন বের করার উপায় হলো ভ্যারিয়েবলের declared অবস্থান দেখা।

মজার ব্যাপার হলো "+" অপারেটর string-এর সাথে কাজ করে। কিন্তু আমরা যা চাই তা সে করে না। String এর জন্য "+" অপারেটর concatenation হিসেবে কাজ করে। অর্থাৎ দুটি operands-কে end-to-end সংযুক্ত করে এই "+" অপারেটর। অর্থাৎ "Hello, " + "world." এর রেজাল্ট হিসেবে আমরা পাবো "Hello, world."।

২.৯ কম্পোজিশন (Composition)

এতক্ষণ পর্যন্ত আমরা প্রোগ্রামিং ল্যাঙ্গুয়েজের (programming language) বিভিন্ন উপকরণ যেমন- variables, expressions এবং statements নিয়ে আলাদা আলাদাভাবে আলোচনা করলাম, কিভাবে তাদের সম্পৃক্ত করা

যায় তা ছাড়াই।

প্রোগ্রামিং ল্যাঙ্গুয়েজের অন্যতম বৈশিষ্ট্য হলো, এক্ষেত্রে আমরা ছোট ছোট ব্লক (block) তৈরি করে তারপর সেগুলোকে কম্পোজ (compose) করতে পারি। যেমন; আমরা জানি কীভাবে দুটি সংখ্যা গুণ করতে হয়, আমরা জানি কীভাবে প্রিন্ট করতে হয়। এই দু'ধরনের কাজকে আমরা যদি একটি একক statement-এর মধ্যে লিখতে চাই তাহলে আমাদের লিখতে হবে:

```
System.out.println(17 * 3);
```

নম্বর, স্ট্রিং এবং ভ্যারিয়েবল সম্বলিত যেকোন expression-ই প্রিন্ট স্টেটমেন্ট (print statement)-এর ভিতরে লিখতে হয়। আমরা পূর্বেই এমন উদাহরণ দেখেছি:

```
System.out.println(hour*60 + minute);
```

আবার আমরা assignment statement-এর ডান দিকে অযৌক্তিক (arbitrary) expressions-ও লিখতে পারি:

```
int percentage;
percentage = (minute * 100) / 60;
```

এধরনের উপস্থাপনা এখন খুব আগ্রহজনক মনে না হলেও, পরে আমরা বিভিন্ন উদাহরণে দেখবো অনেক জটিল গণনার কাজগুলোও composition অনেক সহজে এবং সংক্ষেপে প্রকাশ করে।

সতর্কতা: Assignment-এর বাম দিকের অবশ্যই ভ্যারিয়েবলের নাম থাকবে, expression নয়। কারণ বাম দিকই সংরক্ষণ স্থান নির্দেশ করে যেখানে রেজাল্টগুলো সংরক্ষিত হয়। Expressions কখনোই সংরক্ষণ অবস্থানের প্রতিনিধিত্ব করে না। এটি শুধুমাত্র value এর কথাই প্রকাশ করে। তাই কোডের minute+1 = hour; এরকম প্রকাশ অযৌক্তিক।

২.১০ শব্দকোষ (Glossary)

variable: value সংরক্ষণের নাম অবস্থান হলো variable যার সুনির্দিষ্ট নাম থাকে। সব variable-এর নির্দিষ্ট ধরন আছে যা variable তৈরির সময় declared করতে হয়।

value: নম্বর বা স্ট্রিং বা অন্য যে কোন কিছু (যার নাম পরে দেওয়া হবে) যা variable-এ সংরক্ষিত থাকে তাকেই value বলে। প্রত্যেকটি value-ই কোন না কোন ধরনের হয়।

type: এক সেট value-এ হলো type। variable-এর এই type-ই নির্দেশ করে কোন ধরনের value এখানে সংরক্ষিত হবে। এ পর্যন্ত আমরা যে type-গুলো দেখেছি তা হলো integers (int in Java) and strings (String in Java)

keyword: কিওয়ার্ড হলো কম্পাইলারের জন্য সংরক্ষিত বিশেষ শব্দাবলী যা একটি প্রোগ্রামের মধ্যকার সম্পর্কগুলো সঠিকভাবে কাজ করাতে সাহায্য করে। আমরা public, class এবং void এর মতো কিওয়ার্ডগুলোকে ভ্যারিয়েবলের নাম হিসেবে ব্যবহার করতে পারি না।

declaration: একটি statement যা নতুন ভ্যারিয়েবল তৈরি করে এবং ভ্যারিয়েবলের type নির্দেশ করে।

assignment: একটি statement যা ভ্যারিয়েবলে value assign করে।

expression: Expression হলো ভ্যারিয়েবল, অপারেটর এবং value সম্বলিত রূপ যা একটি একক value-এর প্রতিনিধিত্ব করে। Expression-এরও type রয়েছে যা operators এবং operands দ্বারা নির্ধারিত হয়।

operator: Operator হলো প্রতীক বা চিহ্ন যা যোগ, বিয়োগ, গুণের মতো গণনার কাজ বা স্ট্রিং concatenation কাজ বুঝাতে ব্যবহৃত হয়।

operand: যে value-এর ওপর অপারেটর অপারেট করে তাই operand।

precedence: অপারেশন মূল্যায়নের বা বাস্তবায়নের ক্রমই হলো precedence।

concatenate: দুটি operands - কে end-to-end সংযুক্ত করাই concatenate-এর কাজ।

composition: জটিল statements এবং expressions-এর জটিল হিসেবকে সহজে এবং সংক্ষেপে সহজ expressions এবং statements-এর সমন্বয়ে উপস্থাপন করার ক্ষমতাই composition।

২.১১ অনুশীলনী

অনুশীলনী ২.১: যদি তুমি এই বইটি শ্রেণিকক্ষে পড়ে থাকো তাহলে তোমার একজন বন্ধু খুঁজে নাও এবং তার সাথে "Stump the Chump" এই খেলাটি খেলো।

এই প্রোগ্রামটি শুরু করো এবং সঠিকভাবে কম্পাইল এবং রান করাও। যখন প্রথম খেলোয়াড়ের সময় আসবে তখন দ্বিতীয় খেলোয়াড় সরে যাবে এবং প্রথম খেলোয়াড় প্রোগ্রামে একটি ত্রুটি বা error সংযুক্ত করবে। এরপর দ্বিতীয় খেলোয়াড় সেটি কম্পাইল না করেই সংশোধনের চেষ্টা করবে। যদি দ্বিতীয় খেলোয়াড় কম্পাইল না করেই ত্রুটিটি সংশোধন করতে পারে তাহলে সে দুই পয়েন্ট পাবে আর যদি কম্পাইল করে সংশোধন করে তাহলে প্রথম ও দ্বিতীয় খেলোয়াড় দু'জনেই পাবে এক-এক পয়েন্ট। আর যদি দ্বিতীয় খেলোয়াড় সংশোধন করতে না পারে তাহলে প্রথম খেলোয়াড় পাবে দুই পয়েন্ট।

অনুশীলনী ২.২:

১. একটি নতুন প্রোগ্রাম তৈরি করো এবং এর নাম দাও Date.java। এতে টাইপ করো বা বই থেকে অনুলিপি করে "Hello, World" ধরনের কিছু লিখো এবং তা কম্পাইল ও রান করাও।

২. বই-এর ২.৪ অনুচ্ছেদের উদাহরণ অনুসরণ করে একটি প্রোগ্রাম তৈরি করো যাতে ভ্যারিয়েবলের নাম হবে day, date, month এবং year। day ভ্যারিয়েবল সপ্তাহের দিনের নামগুলো ধারণ করবে এবং date ভ্যারিয়েবল মাসের দিনগুলো ধারণ করবে। প্রত্যেকটি ভ্যারিয়েবলের type কি? এই ভ্যারিয়েবলগুলোতে এমনভাবে value assign করতে হবে যেন তা আজকের তারিখ প্রদর্শন করে।
৩. একটি লাইনে প্রত্যেকটি ভ্যারিয়েবলের value প্রিন্ট করো। এটি এই কাজের মধ্যবর্তী ধাপ, এটি প্রমাণ করবে এই পর্যন্ত তুমি যা যা করেছে তার সবকিছু ঠিক আছে কি নেই।

৪. এই প্রোগ্রামকে এমনভাবে পরিবর্তন করো যেন তা বাংলাদেশে সময় লেখার জন্য যে আদর্শ ফরমেটে

ব্যবহৃত হয় সেভাবে সময় প্রিন্ট করে দেখায়।
form: Saturday, July 16, 2011

৫. আবার এই প্রোগ্রামকে পরিবর্তন করো যেন আউটপুট নিম্নরূপ হয়:
American format:
Saturday, July 16, 2011
European format:
Saturday 16 July, 2011

এই অনুশীলনীতে আমাদের বিভিন্ন ধরনের value প্রদর্শনের জন্য string concatenation ব্যবহার করতে হবে। এছাড়া একই সাথে ছোট ছোট statements যোগ করে ধারাবাহিকভাবে প্রোগ্রাম তৈরির অনুশীলন করতে হবে।

অনুশীলনী ২.৩:

১. একটি নতুন প্রোগ্রাম তৈরি করো এবং এর নাম দাও Time.java। এখন তোমাদের মনে করিয়ে দিতে চাই না যে তোমাকে একটি ছোট প্রোগ্রামের কাজ এখনই শুরু করতে হবে কিন্তু তোমাকে তা অবশ্যই করতে হবে।

২. বই-এর ২.৬ অনুচ্ছেদের উদাহরণ অনুসরণ করে একটি প্রোগ্রাম তৈরি করো যাতে ভ্যারিয়েবলের নাম হবে hour, minute এবং second। এই ভ্যারিয়েবলগুলোর মধ্যে বর্তমান সময়ের value assign করো। ২৪ ঘন্টার সময় ধরে কাজ করো যাতে 2pm-এর স্থলে সময় দেখাবে 14।

৩. এমন একটি প্রোগ্রাম তৈরি করো যেন তা যা মধ্যরাতের আগ পর্যন্ত সেকেন্ড হিসেব করবে এবং প্রিন্ট করে দেখাবে।

৪. এমন একটি প্রোগ্রাম তৈরি করো যেন তা যা মধ্যরাতের পর থেকে দিনের বাকি সময়ের সেকেন্ড হিসেব করবে এবং প্রিন্ট করে দেখাবে।

৫. এমন একটি প্রোগ্রাম তৈরি করো যেন তা যা দিনের শতকরা কত ভাগ পার হয়েছে তা হিসেব করবে এবং প্রিন্ট করে দেখাবে।

৬. hour, minute এবং second-এর value এমনভাবে পরিবর্তন করো যেন তা চলমান সময়কে প্রদর্শন করে এবং পরীক্ষা করে দেখো এই প্রোগ্রাম বিভিন্ন value-তে একইরকমভাবে কাজ করে কিনা।

এই অনুশীলনের মূল উদ্দেশ্য হলো কিছু গাণিতিক অপারেশন ব্যবহার করা। এছাড়াও কিছু জটিল হিসেব যেমন সময় যা অনেকগুলো value নিয়ে কাজ করে তা সম্পর্কে চিন্তা করতে শেখাবে। অবশ্য তোমাদের হয়তো সমস্যা হতে পারে intger(ints)-গুলোর শতকরা গণনা করার সময়, যা পরর্বতী অধ্যায়ের floating point numbers নিয়ে পড়া শুরু করার প্রেষণা হিসেবে কাজ করবে।

ইঙ্গিত : তোমরা অস্থায়ী value ধারণ করার জন্য কিছু সম্পূরক ভ্যারিয়েবল ব্যবহার করতে পারো। এই ধরনের ভ্যারিয়েবলের কাজ হলো গণনাতে সাহায্য করা কিন্তু তা কখনোই প্রিন্ট হবে না। এই ভ্যারিয়েবলগুলো মাঝে মাঝে বলে অস্থায়ী (temporary) বা মধ্যবর্তী (intermediate) ভ্যারিয়েবল।

অধ্যায় ৩

ভয়েড মেথড (Void methods)

৩.১ ফ্লোটিং পয়েন্ট (Floating-point)

পূর্ববর্তী অধ্যায়ে আমরা যে সব নম্বর integer নয় সেই সব নম্বরগুলো নিয়ে কাজ করার সময় কিছু সমস্যায় পড়েছিলাম। আমরা ভগ্নাংশের বদলে অনুপাত পরিমাপ করে এই সমস্যার সমাধান করতে চেয়েছিলাম, কিন্তু এর একটি সাধারণ সমাধান হলো ফ্লোটিং পয়েন্ট (floating-point) ব্যবহার করা। যেটি ভগ্নাংশকে integer হিসেবে উপস্থাপন করতে পারে। জাভায় ফ্লোটিং পয়েন্টকে (floating-point) বলা হয় double, যেটি "double-precision" এর সংক্ষিপ্তরূপ।

তুমি একই সিনট্যাক্স ব্যবহার করে খুব সহজেই floating-point এর ভেরিয়েবল তৈরি করতে পারো এবং অন্য ধরনের একটি মান নির্ধারণ করতে পারো। উদাহরণস্বরূপ:

```
double pi;
pi = 3.14159;
```

একই সময়ে একটি ভেরিয়েবলকে ডিক্লেয়ার করা এবং একটি ভ্যালু নির্ধারণ করা যায় :

```
int x = 1;
String empty = "";
double pi = 3.14159;
```

এই সিনট্যাক্স একটি সাধারণ ঘটনা; একটি সমন্বিত ঘোষণা এবং অ্যাসাইনমেন্টকে অনেক সময় ইনিশিয়ালাইজেশনও (Initialization) বলা হয়।

যদিও floating-point প্রয়োজনীয়, এগুলো সন্দেহের একটি উৎস, কারণ মনে করা হয় এটি পূর্ণসংখ্যা (integer) এবং floating-point এর উপরিলিখন। উদাহরণস্বরূপ; যদি তোমার একটি ভ্যালু 1 থাকে তবে এটি কি integer, floating-point অথবা উভয়ই?

জাভা floating-point মান 1.0 থেকে Integer মান 1 এর পার্থক্য ধরতে পারে, যদিও তাদেরকে একই নম্বর মনে করা হয়। তারা পৃথক ধরনকে নিয়ে আসে এবং খুব কঠিনভাবে বলে যে, তুমি এই ধরনগুলোর কাজের উপযুক্ত নও। উদাহরণস্বরূপ, নিচের কাজগুলো যুক্তিসঙ্গত নয়:

```
int x = 1.1;
```

কারণ বামপাশের ভেরিয়েবল একটি integer এবং ডান পাশেরটি একটি double। কিন্তু এই নিয়মটি ভুলে যাওয়া সহজ, কারণ জাভা স্বয়ংক্রিয়ভাবে এক ধরন থেকে আরেক ধরনে রূপান্তরিত করে থাকে। উদাহরণস্বরূপ:

```
double y = 1;
```

যদিও কারিগরিভাবে এটি আইনত সিদ্ধ না, কিন্তু জাভা অনুমতি দেয় স্বয়ংক্রিয়ভাবে integer থেকে double এ রূপান্তরে। এটি সুবিধাজনক, কিন্তু সমস্যা তৈরি করতে পারে; উদাহরণস্বরূপ :

```
double y = 1 / 3;
```

তুমি অবশ্যই আশা করবে y ভেরিয়েবল 0.333333 মানটি যেন পায়। যেটা একটি সত্যিকারের floating-point value। কিন্তু এটির value দেখায় 0.0। এর কারণ হলো ডানপাশের এক্সপ্রেশনটি হল দুইটি integer এর অনুপাত। তাই জাভা integer এর ভাগ করতে পারে, যেখানে integer হিসেবে 0 ধারণ করে। এটিকে floating-point এ রূপান্তরিত করলে ফলাফল দাড়ায় 0.0।

এই সমস্যা সমাধানের একটি উপায় হলো, ডানপাশের এক্সপ্রেশনকে floating-point পরিনত করা:

```
double y = 1.0 / 3.0;
```

এখানে y কে 0.333333 এ নির্ধারণ করে দেওয়া হয়েছে।

অপারেশন হিসেবে আমরা যা দেখে থাকি অর্থাৎ যোগ, বিয়োগ, গুণ, ভাগ এর সবই floating-point এর সাথে কাজ করে। তুমি হয়তো জেনে আগ্রহী হবে যে, underlying মেকানিজম সম্পূর্ণ পৃথক। যা হোক, অনেক প্রসেসরের বিশেষ হার্ডওয়ার থাকে শুধু মাত্র floating-point এর অপারেশন করার জন্য।

৩.২ ডাবল থেকে পূর্ণসংখ্যায় রূপান্তর

আমি বলেছিলাম যে, প্রয়োজনে জাভা int থেকে double সংখ্যায় স্বয়ংক্রিয়ভাবে পরিবর্তন হয়ে থাকে, কারণ এতে কোন তথ্য হারানো যায়না। অন্যদিকে double থেকে int এ রূপান্তরে রাউন্ডিং অফ এর দরকার পড়ে। জাভা স্বয়ংক্রিয়ভাবে এই ধরনের অপারেশন চালাতে পারে না তাই একজন প্রোগ্রামার হিসেবে floating-point এর কোন ভ্যালু হারানো গেল কিনা তার ব্যাপারে সচেতন থাকতে হবে যে।

floating-point থেকে intger এ রূপান্তরিত করার একটি সহজতম উপায় হলো টাইপকাস্ট (typecast) ব্যবহার করা। টাইপকাস্টকে বেশির ভাগ সময় কল করা হয় কারণ, এটা এক ধরনের ভ্যালু নিয়ে তাকে অন্য ধরনে "পরিনত" করে (কাস্ট করে) এটা অনেকটা মলডিং ও রিফর্মিং এর মতো।

টাইপকাস্টিং এর জন্য সিনটেক্সের ব্র্যাকেটে এর ধরনের নাম রাখা হয় এবং এটি অপারেটর হিসাবে ব্যবহার করা হয়। উদাহরণস্বরূপ:

```
double pi = 3.14159;
int x = (int) pi;
```

(int) অপারেটর রূপান্তরের একটি প্রভাব যা একটি interger এ পরিনত করে। তাই x মান হিসেবে পায় 3।

টাইপকাস্টিং গাণিতিক অপারেটরের চেয়ে অগ্রাধিকার পায়, তাই নিচের উদাহরণে আমরা দেখতে পাই, রূপান্তরের সময় সবার আগে পাইয়ের মান পূর্ণ সংখ্যায় রূপান্তরিত হয় এবং ফলাফল দেখায় 60.00, যদিও ফলাফল হওয়ার কথা 62।

```
double pi = 3.14159;
double x = (int) pi * 20.0;
```

interger এ রূপান্তরের সময় সংখ্যাটি সবসময় পূর্ণ সংখ্যায় হয়, এমনকি ভগ্নাংশটি যদি 0.99999999 হয়। এই আচরণটি (অগ্রাধিকার ও পূর্ণসংখ্যা) টাইপকাস্টিংকে মাঝে মাঝে ত্রুটিপ্রবণ করে তুলে।

৩.৩ ম্যাথ মেথডস

সম্ভবত গণিতে তোমরা কিছু ফাংশন যেমন সাইন এবং লগ ইত্যাদি দেখে থাকবে এবং এই সব এক্সপ্রেশন দিয়ে পরিমাপ করতে শিখে থাকবে যেমন- sin(π /2) এবং log(1/x)। প্রথমে তুমি বন্ধনীর ভেতরের এক্সপ্রেশনের কাজ করবে, যাকে বলা হয় ফাংশনের আর্গুমেন্ট। বিভিন্ন টেবিলের মধ্য খুঁজে অথবা বিভিন্ন কার্য সম্পাদনের মাধ্যমে তুমি ফাংশনটার মান নির্ণয় করতে পারবে।

উপরের কাজটি বারে বারে প্রয়োগ করা যাবে যদি আরও জটিল এক্সপ্রেশন আসে, যেমন log(1/sin(π /2))। এই এক্সপ্রেশনে প্রথমে আমরা প্রথম ফাংশনের আর্গুমেন্ট নিয়ে কাজ করবো, পরে ফাংশন নিয়ে কাজ করতে হবে এবং পরে অন্য কিছু নিয়ে।

যেসব সাধারণ গাণিতিক অপারেশনগুলো বেশিরভাগ সময় ব্যবহার করা হয় সেগুলো জাভায়ও ব্যবহার করা হয়। এই ফাংশনগুলোকে মেথডস বলা হয়। আমরা আগেই দেখেছি, গাণিতিক ম্যাথড একটি সিনট্যাক্স ব্যবহার করে যা প্রিন্ট স্টেটটমেন্টের মতোই:

```
double root = Math.sqrt(17.0);
double angle = 1.5;
double height = Math.sin(angle);
```

প্রথম উদাহরণে 17.0 এর বর্গমূলকে বর্গমূল হিসেবে নির্ধারণ করা হয়েছে। দ্বিতীয় উদাহরণে আমরা দেখি ত্রিভূজের তিনকোনের মান নির্ধারণ করা হয়েছে, যা 1.5। জাভা ধরে নেয় যে, আমরা সাইন বা অন্যান্য জ্যামিতিক ফাংশনে (cos, ten) যে যে মান ব্যবহার করি তা রেডিয়ান। ডিগ্রি থেকে রেডিয়ানে কনভার্ট করতে তোমাকে ৩৬০ দিয়ে ভাগ করতে হবে এবং তার সাথে 2π গুণ করতে হয়। জাভা math.PI যোগান দেয় যেমন:

```
double degrees = 90;
double angle = degrees * 2 * Math.PI / 360.0;
```

নোট: PI সর্বদা বড় হাতের অক্ষরে হবে। জাভা Pi, pi, pie বুঝতে পারে না।

ম্যাথ ক্লাশের আরেকটি দরকারি মেথড হলো রাউন্ড, যেটা floating-point কে তার নিকটবর্তী integer এ পরিনত করে, এবং মান হিসেবে **int** প্রদান করে।

```
int x = Math.round(Math.PI * 20.0);
```

এই ক্ষেত্রে আমরা দেখতে পাই, মেথড ব্যবহারের পূর্বে, প্রথমে গুণের কাজ হবে। ফলাফল পাওয়া যাবে 63 (62.8319 এর পূর্ণ সংখ্যা)।

৩.৪ কমপোজিশন Composition

গাণিতিক ফাংশনের মতো, জাভার মেথডগুলো একত্রে ব্যবহার করা যায়, অর্থাৎ তুমি একটি এক্সপ্রেশনকে অন্য একটি এক্সপ্রেশনের অংশ হিসেবে ব্যবহার করতে পারো। উদাহরণস্বরূপ, তুমি যেকোন এক্সপ্রেশনকে একটি মেথডের আর্গুমেন্ট হিসেবে ব্যবহার করতে পারো:

```
double x = Math.cos(angle + Math.PI/2);
```

এই স্ট্যাটমেন্ট math.PI এর মান নিয়ে তাকে দুই দিয়ে ভাগ করে এবং এবং এর সাথে চলকের কোনের পরিমাপ যোগ করে। এই যোগফল cos এর মান হিসাবে দেখায়। (PI এখানে চলকের নাম, মেথডের নাম নয়, তাই এখানে কোন আর্গুমেন্ট নাই, এমনকি কোন খালি আর্গুমেন্ট () ও নেই)।

তুমি একটি মেথডের ফলাফল নিয়ে তা অন্য একটি আর্গুমেন্ট হিসাবে ব্যবহার করতে পারো।

```
double x = Math.exp(Math.log(10.0));
```

জাভায় লগ মেথডে সর্বদাই base e ব্যবহার করা হয়, তাই এই স্ট্যাটমেন্ট log base e of 10 খুঁজে এবং এরপর এটি e কে এই পাওয়ারে উন্নীত করে। ফলাফল হিসেবে x নির্ধারণ করা আছে; আমি আশা করি এটি কি হবে তা তুমি বুঝতে পারবে।

৩.৫ নতুন মেথড যোগ করা

যদিও আমরা জাভার লাইব্রেরী থেকেই মেথড ব্যবহার করছি, কিন্তু এখানে নতুন মেথড যোগ করার সম্ভব। আমরা ইতিমধ্যেই একটি মেথড দেখেছি: main। মেথড হিসেবে main নামটি একটি বিশেষ নাম কিন্তু এর সিনট্যাক্স অন্যান্য মেথডের মতোই:

```
public static void NAME( LIST OF PARAMETERS ) {
        STATEMENTS
}
```

তুমি তোমার তৈরিকৃত মেথডের যে কোন নাম দিতে পারো, শুধুমাত্র তুমি এর নাম main দিতে পারবে না বা অন্য কোন জাভা কীওয়ার্ড দিতে পারবে না। রীতি অনুযায়ী জাভা মেথড শুরু হয় একটি ছোট হাতের অক্ষর দ্বারা এবং “camel caps” ব্যবহার করা হয়, যেটা jammingWordsTogetherLikeThis এর একটি সুন্দর নাম।

প্যারামিটারের তালিকা ঠিক করে দেয় নতুন মেথডে কোন ধরনের তালিকা প্রয়োজন, যদি প্রয়োজন পড়ে তাহলে তোমাকে নতুন মেথড ব্যবহার করতে হবে।

main এর জন্য প্যারামিটার হলো string[] args, এর মানে হচ্ছে যে কেউ main ব্যবহার করুক না কেন, তা strings এর অ্যারে কে সরবরাহ করে (আমরা ১২ অধ্যায়ে অ্যারে সম্পর্কে জানতে পারবো)। প্রথম মেথডে আমরা কোন প্যারামিটার ব্যবহার করি নাই, তাই এর সিনট্যাক্স হয়েছে নিম্নরূপ:

```
public static void newLine() {
        System.out.println("");
}
```

এই মেথডের নাম newLine, এবং খালি বন্ধনী দ্বারা বোঝায় এটি কোন প্যারামিটার ধারণ করে না। এটি একটি মাত্র স্ট্যাটমেন্ট ধারণ করে, যেটা একটি খালি স্ট্রিং "" প্রিন্ট করে। কোন অক্ষর ছাড়া একটি স্ট্রিং প্রিন্ট করলে তা সব সময় দরকারী নাও হতে পারে, কিন্তু println কোন কিছু প্রিন্ট করেই পরবর্তী লাইনে চলে যায়, তাই এই স্ট্যাটমেন্ট পরবর্তী লাইনে চলে যাবে। main এ আমরা এই নতুন মেথডকে কাজে লাগাতে পারি, আবার জাভাতেও একই ভাবে এটিকে ব্যবহার করতে পারি:

```
public static void main(String[] args) {
        System.out.println("First line.");
        newLine();
        System.out.println("Second line.");
}
```

এই প্রোগ্রামারের আউটপুট হবে নিম্নরূপ:

First line.

Second line.

এই দুটি বাক্যের মাঝখানের ফাঁকা স্থানের দিকে লক্ষ্য কর, যদি দুটি লাইনের মাঝে আরও ফাঁকা স্থান প্রয়োজন পড়ে তাহলে এই মেথডটি পুনরায় ব্যবহার করতে পারি:

```
public static void main(String[] args) {
        System.out.println("First line.");
        newLine();
        newLine();
        newLine();
        System.out.println("Second line.");
}
```

অথবা আমরা একটি নতুন মেথড লিখতে পারি, যার নাম হবে threeLine, যেটি নতুন তিনটি লাইন প্রিন্ট করবে:

```
public static void threeLine() {
        newLine(); newLine(); newLine();
}

public static void main(String[] args) {
        System.out.println("First line.");
        threeLine();
        System.out.println("Second line.");
}
```

এই প্রোগ্রাম থেকে আমরা কয়েকটি বিষয় দেখতে পাই:

- তুমি সহজেই একই প্রণালী একাধিকবার ব্যবহার করতে পারবে।
- তুমি খুব সহজেই একটি মেথডকে আরেকটি মেথডে ব্যবহার করতে পারো, এই ক্ষেত্রে, main আহ্বান (invoke) করেছে threeLine এবং threeLine আহ্বান করেছে newLine.
- threeLine এ তিনটি লাইন লেখা হয়েছে যা সবই একই লাইন। এটা সিনটেকটিক্যালি সমর্থিত (নম্বর হিসাবে যে খালি স্থান ও নতুন লাইন প্রোগ্রামে কোন পরিবর্তন করে না)। তবে এটি একটি ভালো চিন্তা যে প্রতিটি স্ট্যাটমেন্ট তার নিজের লাইনে রাখা, তবে এই বইয়ে মাঝে মাঝে আইনটি ভাঙ্গা হয়েছে।

নতুন মেথডগুলো তৈরি করা কেন কষ্টকর তা জেনে তুমি অবশ্যই আশ্চর্য হবে। এর অনেক কারণ আছে; এখানে দুইটা উদাহরণ দেওয়া হল:

1. একটি নতুন মেথড তৈরি করলে তা একদল স্ট্যাটমেন্টের একটি নতুন নাম দেওয়ার সুযোগ সৃষ্টি করে। একটি মাত্র স্ট্যাটমেন্টের আড়ালে জটিল হিসাব লুকিয়ে রেখে মেথড একটি প্রোগ্রামকে সহজ করতে পারে এবং এবং এটি করা হয় গোপন সংকেতের বদলে ইংরেজী শব্দ ব্যবহার করে। যেমন কোনটা বোঝা বেশি সহজ? newLine নাকি system.out.println(“”)।

2. একটি নতুন মেথড তৈরি তা পুনরাবৃত্তিমূলক কোডের ব্যবহারের বদলে প্রোগ্রামকে অনেক ছোট করে নিয়ে আসে, উদাহরণস্বরূপ, ধারাবাহিকভাবে নয়টি নতুন লাইন প্রিন্ট করতে চাইলে তুমি সহজেই তিনবার threeLine ব্যবহার করতে পারো।

আমরা ৭.৬ নম্বর সেকশনে আবারও এই প্রশ্ন নিয়ে ফিরে আসবো এবং একটি প্রোগ্রামকে কিছু মেথডে ভাগ করে করার সুবিধাগুলোর তালিকা তৈরি করবো ।

৩.৬ ক্লাশ এবং মেথড (classes and methods)

আগের অধ্যায়ের কোডগুলোর ফ্র্যাগমেন্ট একসাথে করলে class কে আমরা দেখতে পাই নিচের মতো:

```
class NewLine {

        public static void newLine() {
                System.out.println("");
        }

        public static void threeLine() {
                newLine(); newLine(); newLine();
        }

        public static void main(String[] args) {
                System.out.println("First line.");
                threeLine();
                System.out.println("Second line.");
        }
}
```

প্রথম লাইন নির্দেশ করে যে, নতুন ক্লাশ newLine এর জন্য এটি একটি ক্লাশ ডেফিনেশন। একটি class হল সম্পর্কযুক্ত মেথডগুলোর সংগ্রহশালা। এই ক্ষেত্রে NewLine নামের class টি তিনটি মেথডকে ধারণ করে। এগুলো হল- newLine, threeLine এবং main.

আরেকটি class আমরা দেখতে পাই math class এ। এটি sqrt, sin এবং আরও কিছু নামের মেথডকে ধারণ করে। আমরা যখন একটি গাণিতিক মেথডকে ব্যবহার করি, তখন আমরা math class এর নাম এবং মেথডের নাম০ নির্দিষ্ট করে দেই। আর তাই জাভায় মেথডের সিনট্যাক্সটা একটু পৃথক। মেথড হিসেবে আমরা লিখি:

```
Math.pow(2.0, 10.0);
newLine();
```

প্রথম স্ট্যেটমেন্টে math class এ pow মেথডটি ব্যবহার করা হয়েছে (যেটা প্রথম আর্গুমেন্টকে দ্বিতীয় আর্গুমেন্টের ঘাত হিসাবে উত্তোলন করা হয়েছে)। দ্বিতীয় স্ট্যেটমেন্টে newLine মেথডকে ব্যবহার করা হয়েছে। যেটাকে জাভা অনুমান করে ক্লাশ হিসেবে। (যেমন newLine)।

তুমি যদি একটি ভুল ক্লাশকে মেথড হিসেবে ব্যবহার করতে চাও তাহলে কম্পাইলার এটিকে ভুল হিসাবে দেখাবে। উদাহরণস্বরূপ যদি তুমি টাইপ কর:

```
pow(2.0, 10.0);
```

কম্পাইলার এই জাতীয় কিছু বলবে, “Can't find a method named pow in class NewLine.” । তুমি যদি এই জাতীয় বার্তা দেখতে পাও, তাহলে অবাক হতে যে, এটি কেন তোমার ক্লাশ ডেফিনেশনে pow এর মতো, এখন তুমি এটির কারণ জানো।

৩.৭ একাধিক মেথডের সাথে প্রোগ্রামিং

একাধিক মেথড ধারণকারী একটি ক্লাশের সংজ্ঞা যখন তুমি দেখবে, এটি তখন উপর থেকে নিচের দিকে পরীক্ষা করতে শুরু করবে, কিন্তু এটা একটি বিভ্রান্তিকর অবস্থা। কারণ এটি প্রোগ্রাম সম্পাদনার সঠিক বিন্যাস নয়।

একটি প্রোগ্রামে main স্ট্যাটমেন্ট কোথায় আছে তা বিবেচনা না করেই সবার আগে তার সম্পাদন শুরু হয় (এই উদাহরণে আমি এটি সবার উপরে রেখেছি)।

একই সময়ে একটি মাত্র স্ট্যাটমেন্ট নির্বাহ হয়, ধারাবাহিকভাবে, যতক্ষণ পর্যন্ত না একটি মেথডে পৌঁছানো হয়। মেথডে পৌঁছানো হল প্রোগ্রাম নির্বাহের বিকল্প পথে চলা। পরবর্তী স্ট্যেটমেন্টে যাওয়ার পরিবর্তে, তুমি ব্যবহৃত মেথডের প্রথম লাইনে যেতে পারো। এখানে সকল স্ট্যাটমেন্ট সম্পাদন করে তুমি ফিরে আসতে পারো এবং যেখান থেকে তুমি কাজ শেষ না করে প্রথম লাইনে গিয়েছিলে সেখান থেকে কাজ শুরু করতে পারো।

এটা বলা খুব সহজ, তবে এর ব্যতিক্রম যে, শুধু তোমার মনে রাখতে হবে, একটি মেথড অন্য মেথডেও ব্যবহার করা যায়। এইভাবে, যতক্ষণ পর্যন্ত না তুমি main এর মাঝে আছো, আমাদের অবশ্যই threeLine স্ট্যেটমেন্টে যেতে হবে এবং সম্পাদন করতে হবে। কিন্তু যখন আমরা threeLine সম্পাদন করবো, আমরা তিনবার বাধাহীন ভাবে বন্ধ করতে পারবো এবং newLine সম্পাদন করতে পারবো।

এর অংশ হিসাবে, অন্য একটি বিকল্পের কারণে println newLine তৈরি করে। ভাগ্যক্রমে, প্রোগ্রামটি কোন অবস্থায় আছে তার চিহ্ন জাভা ধারণ করে রাখতে পারে। তাই যখন println এর কাজ শেষ হয়, তখন এটা যেখান থেকে newLine এর কাজ বন্ধ ছিল সেখানে ফিরে যেতে পারে, এবং পরে threeLine ফিরে পায়, এবং এরপর শেষপর্যন্ত এটি যেখান থেকে প্রোগ্রামটি ক্ষান্ত হয়েছিল সেই main এ ফিরে যায়।

টেকনিক্যালি, main এর কাজ শেষ হওয়ার আগ পর্যন্ত প্রোগ্রাম শেষ হয় না। জাভা ইন্টারপ্রেটারে যখন একটি প্রোগ্রাম শেষ হয় তখন এটি main এর কাজ শুরু করে। ইন্টারপ্রেটার কিছু কাজ করে যেমন-উইন্ডো মুছে ফেলা, সাধারণ পরিষ্কার করা এবং পরে প্রোগ্রাম সমাপ্ত করা।

এর মানে কি দাঁড়াল? তুমি যখন একটি প্রোগ্রাম পড়বে তখন কখনও উপর থেকে পড়ে নিচের দিকে আসবে না। এর পরিবর্তে, প্রোগ্রাম সম্পাদনের ধারাবাহিকতাকে অনুসরণ কর।

৩.৮ প্যারামিটার এবং আর্গুমেন্ট (Parameters and arguments)

কিছু মেথডে আমরা চাহিদা মতো আর্গুমেন্ট ব্যবহার করি, যার মান আমরা যখন এই মেথডগুলো তৈরি করি তখনই দেই। উদাহরণস্বরূপ, কোন নম্বরের সাইন (sine) এর মান বের করতে, আমাদেরকে নম্বরটি সরবরাহ করতে হবে। তাই sin আর্গুমেন্ট হিসেবে ডাবল নিবে। একটি স্ট্রিং প্রিন্ট করতে আমাদের স্ট্রিং টি দিতে হবে, তাই বলা যায় println আর্গুমেন্ট হিসেবে একটি স্ট্রিং নেয়।

কিছু মেথড একাধিক আর্গুমেন্ট নেয়; উদাহরণস্বরূপ, pow দুইটি double নেয়, একটি base এবং অপরটি exponent ।

যখন তুমি একটি মেথড ব্যবহার কর, তুমি তার জন্য একটি আর্গুমেন্ট প্রদান করতে হয়। যখন তুমি একটি মেথড লেখ। তোমাকে এর জন্য একটি প্যারামিটারের তালিকা নির্দিষ্ট করতে হয়। একটি প্যারামিটার হল একটি ভেরিয়েবল যেখানে একটি আর্গুমেন্ট সংরক্ষিত থাকে। কোন আর্গুমেন্টটা প্রয়োজন তা প্যারামিটারের তালিকা নির্দিষ্ট করে থাকে।

উদাহরণস্বরূপ, printTwice একটি একক প্যারামিটার 's' কে নির্দেশ করে, যেটির ধরন হল স্ট্রিং। আমি এটিকে s নামে ডাকার পরামর্শ দেব, কারণ এটি একটি স্ট্রিং, কিন্তু আমি এটার কোন যৌক্তিক নাম দিতে পারি।

```
public static void printTwice(String s) {
        System.out.println(s);
        System.out.println(s);
}
```

যখন আমরা printTwice ব্যবহার করবো তখন টাইপস্ট্রিং এর সাথে একটি একক আর্গুমেন্টও দিয়ে দেব।

```
printTwice("Don't make me say this twice!");
```

যখন তুমি একটি মেথড ব্যবহার করবে, তখন যে আর্গুমেন্ট সরবরাহ করবে তার প্যারামিটারও তোমাকে নির্দিষ্ট করতে হবে। এই উদাহরণে, আর্গুমেন্ট "Don't make me say this twice!" এর জন্য প্যারামিটার নির্দিষ্ট করা হয়েছে s। এই প্রক্রিয়াকে বলে প্যারামিটার পাসিং (parameter passing) কারণ বাইরে থেকে মান নিয়ে এই মেথডটি ভেতরে পাঠিয়ে দেয়।

একটি আর্গুমেন্ট যে কোন প্রকারের এক্সপ্রেশন হতে পারে। তাই তোমার যদি একটি স্ট্রিং ভেরিয়েবল থাকে, তুমি এটিকে আর্গুমেন্ট হিসেবে ব্যবহার করতে পারো:

```
String argument = "Never say never.";
printTwice(argument);
```

অবশ্যই তুমি আর্গুমেন্ট হিসাবে যে মান দিবে তার ধরন ও প্যারামিটারের ধরন একই হতে হবে। উদাহরণস্বরূপ, তুমি যদি এই রকম লিখে থাকো

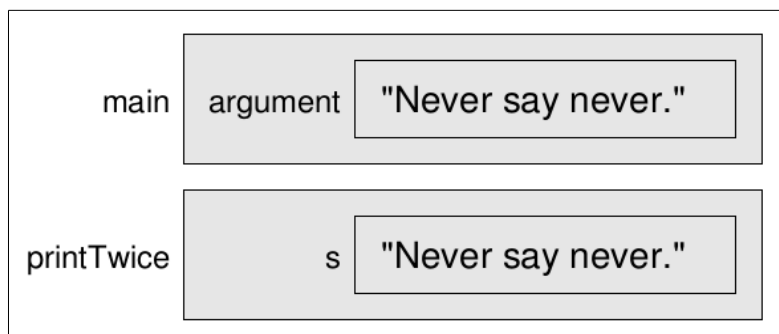
```
printTwice(17);
```

তাহলে একটি এরর বার্তা দেখতে পারবে, “cannot find symbol," যেটা খুব বেশি সাহায্যকারী নয়। এর কারণ হল, জাভা printTwice নামে একটি মেথড খুঁজছে, যেটা একটি পূর্ণসংখ্যা আর্গুমেন্ট হিসাবে ব্যবহার করা হয়েছে। যেহেতু তারা এক নয়, এটি “symbol” এর মতো কিছু খুঁজে পাবে না।

system.out.println যে কোন ধরনের আর্গুমেন্টকে গ্রহণ করতে পারে। কিন্তু এটা একটি ব্যতিক্রম; বেশির ভাগ মেথড অমায়িক নয়।

৩.৯ স্ট্যাক ডায়াগ্রাম (Stack diagrams)

প্যারামিটার এবং অন্যান্য ভেরিয়বলগুলো শুধু মাত্র তাদের ভেতরের মেথডে বিদ্যামান থাকে। main এর ভেতর সীমাবদ্ধ, s এর মতো আর কিছু নেই। তুমি যদি এটা ব্যবহার করতে চেষ্টা কর, কম্পাইলার অভিযোগ জানাবে, অনুরূপভাবে, printTwice এর ভেতরে আর্গুমেন্ট এর মতো আর কিছু নেই।

প্রতিটি ভেরিয়েবল কোথায় নির্ধারন করা হয়েছে তা অনুসরণের জন্য চিহ্ন রাখার একটি উপায় হলো stack diagram। পূর্ববর্তী উদাহরণের stack diagram হল:

প্রতিটি মেথডে একটি ধূসর বক্স আছে যাদের বলা হয় frame যেটা মেথডের প্যারামিটার এবং ভেরিয়েলগুলো ধারণ করে। বাইরের frame এ যেটি দেখা যায় তার নাম মেথড। সাধারণভাবে প্রতিটি ভেরিয়েবলের মান ও এর সাথে সাথের ভেরিয়েবলের সাথে বক্সের নিচে নামে।

৩.১০ একাধিক প্যারামিটারের সাথে মেথড

একাধিক প্যারামিটারের সাথে মেথড ডিক্লেয়ার করা এবং ব্যবহার করার জন্য যে সিনট্যাক্স ব্যবহার করা হয় তা প্রোগ্রামে এরর হওয়ার একটি সাধারণক্ষেত্র। প্রথমে মনে রাখতে হবে যে, আমাকে প্রতিটি প্যারামিটারকে ডিক্লেয়ার করতে হবে। উদাহরণস্বরূপ:

```
public static void printTime(int hour, int minute) {
        System.out.print(hour);
        System.out.print(":");
        System.out.println(minute);
}
```

পূর্ণসংখ্যা হিসেবে hour, minute লেখায় প্রলুব্ধ করা হয়, কিন্তু এই ফরমেটটা শুধু মাত্র ভেরিয়েবল ডিক্লায়ের জন্য প্রযোজ্য, প্যারামিটারের তালিকার জন্য না।

আরেকটি দ্বিধান্বিত হওয়ার সাধারণ উৎস হল যে, তুমি কখনও আর্গুমেন্টের টাইপ ডিক্লেয়ার করতে পারো না। নিচের লাইনগুলো ভুল!

```
int hour = 11;
int minute = 59;
printTime(int hour, int minute);   // WRONG!
```

এই ক্ষেত্রে, জাভা hour এবং minute এর ডিক্লেয়ারেশন দেখে এদের টাইপ কি তা বলতে পারে। যখন তুমি এদেরকে একটি আর্গুমেন্ট হিসাবে চালাবে তখন এটার টাইপ সংযুক্ত করা তেমন দরকার পড়বে না। তাই সঠিক সিনট্যাক্সটি হল printTime(hour, minute)।

৩.১১ যেসব মেথড মান ফেরত দেয়

কিছু মেথড যেমন math আমরা ব্যবহার করি যেগুলো আমাদেরকে মান রিটার্ন করে। অন্যান্য মেথডগুলো যেমন println এবং newLine একটি একশন হিসাবে কাজ করে, কিন্তু তারা কোন মান রিটার্ন করে না। এই বিষয়টি কিছু প্রশ্নের উদ্রেগ করে:

- কি হবে যদি তুমি একটি মেথড তৈরি কর এবং তুমি এর ফলাফলের সাথে কিছু না কর (যেমন-তুমি এটাকে কোন ভেরিয়েবল হিসাবে নির্দিষ্ট করনি অথবা এটিকে তুমি কোন বড় এক্সপ্রেশনের অংশ হিসাবে ব্যবহার করনি)?
- কি হবে যদি তুমি print মেথডকে একটি এক্সপ্রেশনের অংশ হিসাবে ব্যবহার কর, যেমন System.out.println("boo!") + 7 ?
- আমরা কি এমন মেথড ব্যবহার করতে পারি যা মান রিটার্ন করে, অথবা আমরা কি newLine এবং printTwice এর সাথে জড়িত থাকতে পারি?

তৃতীয় প্রশ্নের উত্তর হচ্ছে “আমরা এমন মেথড ব্যবহার করতে পারি যা মান রিটার্ন করে,” এবং আমরা দেখতে পারি একজোড়া অধ্যায় কি করে। অন্য দুইটি প্রশ্নের উত্তর খোঁজার দায়িত্ব আমি তোমাকে দিলাম, তুমি চেষ্টা কর উত্তর

বের করার। আসলে, জাভাতে লিগাল বা ইলিগাল কি তার সম্পর্কে যে কোন সময় তোমার প্রশ্ন থাকতে পারে, এর উত্তর বের করার ভালো উপায় হলো কম্পাইলারকে জিজ্ঞাসা করা।

৩.১২ শব্দকোষ

ইনিশিয়ালাইজেশন: একটি স্ট্যাটমেন্ট যা নতুন ভেরিয়েবল ডিক্লেয়ার করে এবং একই সাথে একটি মান নির্দিষ্ট করে।

ফ্লোটিং পয়েন্ট: এক ধরনের ভেরিয়েবল (অথবা ভ্যালু) যা পূর্ণসংখ্যা হিসাবে ভগ্নাংশ ধারণ করে। ফ্লোটিং পয়েন্ট টাইপ আমরা ডাবল হিসাবে ব্যবহার করতে পারি।

ক্লাশ: মেথডের সংগ্রহশালাকে ক্লাশ নামে ডাকা হয়। আমরা math ক্লাশ এবং system ক্লাশ ব্যবহার করতে পারি এবং আমরা ক্লাশের নাম লিখতে পারি hello এবং NewLine।

মেথড: স্ট্যাটমেন্টের ধারাবাহিক নাম যা দরকারি ফাংশন হিসাবে কাজ করে। মেথড প্যারামিটার নিতেও পারে নাও নিতে পারে এবং এটি ভ্যালু রিটার্নও করতে পারে আবার নাও করতে পারে।

প্যারামিটার: একটি মেথডকে রান করানোর জন্য কিছু সুনির্দিষ্ট তথ্যের প্রয়োজন পড়ে। প্যারামিটার হল ভ্যারিয়েবল: এগুলো ভ্যালু এবং টাইপ ধারণ করে।

আর্গুমেন্ট: আর্গুমেন্ট একটি ভ্যালু যা তুমি যখন একটি মেথড ব্যবহার করো তখন এটিকে সরবরাহ করো। এই ভ্যালুটি অবশ্যই তুমি যে প্যারামিটারটি দিয়েছো তার মতো বা একই ধরনের হতে হবে।

ফ্রেম: একটি কাঠামো (একটি স্ট্যাক ডায়াগ্রামের মধ্যে ধূসর বর্ণের বক্স দ্বারা প্রতিনিধিত্ব করা) যেটা একটি মেথডের প্যারামিটার এবং ভেরিয়েবলকে ধারণ করে।

ইনভক: একটি মেথড কাজ করার কারণ।

৩.১৩ অনুশীলনী:

অনুশীলনী ৩.১: একটি স্ট্যাক ফ্রেম অংকন কর, যেটা সেকশন ৩.১০ এর প্রোগ্রামের অবস্থা বর্ণনা করবে, যখন main, 11 এবং 59 আর্গুমেন্টের সাথে printTime ব্যবহার করবে।

অনুশীলনী ৩.২: এই অনুশীলনীর উদ্দেশ্য হল কোড পড়ার চর্চা করা এবং কিভাবে একাধিক মেথড সমৃদ্ধ একটি প্রোগ্রাম কাজ করে তা বুঝতে পারছো কিনা তা নিশ্চিত করা।

1. নিম্নলিখিত প্রোগ্রামের আউটপুট কি হবে? কোথায় স্পেস হবে এবং কোথায় নতুন লাইন হবে তার সম্পর্কে নির্দেশ কর।

ইঙ্গিত: ping এবং baffle যোগ করার ফলে কি হল তা বর্ণনা করা থেকে শুরু কর।

2. একটি স্ট্যাক ডায়াগ্রাম আঁক যেটা প্রথমবার প্রোগ্রামে ping যুক্ত করলে কি অবস্থা হয় তা দেখাবে।

```
public static void zoop() {
baffle();
System.out.print("You wugga ");
baffle();
}

public static void main(String[] args) {
System.out.print("No, I ");
zoop();
System.out.print("I ");
baffle();
}

public static void baffle() {
System.out.print("wug");
ping();
}
public static void ping() {
System.out.println(".");
}
```

অনুশীলনী ৩.৩: এই অনুশীলনীর উদ্দেশ্য হল, কিভাবে মেথড যেটি প্যারামিটার গ্রহণ করে, সেটি তৈরি এবং ব্যবহার করতে হয় তুমি বুঝতে পেরেছো কিনা তা জানা।

1. zool নামে একটি মেথডের প্রথম লাইন লেখ, যেটা তিনটি প্যারামিটার গ্রহণ করে: একটি পূর্ণসংখ্যা দুইটি স্ট্রিং।
2. zool এর জন্য এক লাইন কোড লিখ, আর্গুমেন্ট হিসাবে ভ্যালু দাও 11, তোমার প্রথম পোষা প্রাণী এবং তুমি যেখানে বড় হয়েছো তার নাম।

অনুশীলনী ৩.৪: এই অনুশীলনীর উদ্দেশ্য হল, আগের অনুশীলনী থেকে কোড নিয়ে একটি মেথড যেটি প্যারামিটার গ্রহণ করে তা এনক্যাপসুলেট করা। তুমি অনুশীলনী ২.২ এর সমাধান থেকে কাজটি শুরু করতে পারো।

1. printAmerican নামে একটি মেথড তৈরি কর, যেটা দিন, তারিখ, মাস এবং বছরকে প্যারামিটার হিসাবে গ্রহণ করে এবং তাদের আমেরিকান ফরমেটে print করে।
2. তোমার মেথডটিতে main ব্যবহার করে তা পরীক্ষা কর এবং উপযুক্ত আর্গুমেন্ট দাও। তোমার আউটপুট নিচের মতো কিছু একটা হবে (শুধুমাত্র তারিখটি ভিন্ন হতে পারে):

Saturday, July 16, 2011

3. এখন তুমি printAmerican টি ডিবাগ করাও এবং আরেকটি মেথড printEuropean ব্যবহার কর যেটা তারিখকে ইউরোপিয়ান ফরমেটে প্রকাশ করবে।

অধ্যায় ৪

নিয়মাবলী এবং পুনরাবৃত্তি

৪.১ মড্যুলাস নিয়ন্ত্রক (The modulus operator)

মড্যুলাস নিয়ন্ত্রক পূর্ণ সংখ্যার (পূর্ণ সংখ্যা প্রকাশ) উপর কাজ করে এবং অবশিষ্টাংশ প্রস্তুত করে যখন প্রথম 'অপারেন্ড' কে দ্বিতীয়টি দ্বারা ভাগ করা হয়। “java”-য় মড্যুলাস নিয়ন্ত্রকটি একটি % চিহ্ন। নিয়মটি অন্যান্য নিয়ন্ত্রকের জন্যও অনুরূপ

```
int quotient = 7 / 3;
int remainder = 7 % 3;
```

প্রথম নিয়ন্ত্রকটি; পূর্ণসংখ্যার বিভাজন 2 প্রস্তুত করে। দ্বিতীয় নিয়ন্ত্রকটি 1 প্রস্তুত করে। অর্থাৎ, 7 কে 3 দ্বারা ভাগ করলে ফলাফল 2, অবশিষ্টাংশ 1।

মড্যুলাস নিয়ন্ত্রক আশ্চর্যজনকভাবে কার্যকরী, উদাহরণস্বরূপ, একটি সংখ্যা আরেকটি দ্বারা বিভাজনসাধ্য কিনা তা তুমি নির্ধারণ করতে পারবে। যদি x % y এর ফলাফল '0' (শূন্য), তাহলে x, y দ্বারা বিভাজনসাধ্য।

এছাড়াও তুমি মড্যুলাস নিয়ন্ত্রককে সর্বোচ্চ সঠিক সংখ্যাটি নির্বাচন করতে ব্যবহার করতে পার। উদাহরণস্বরূপ, x %10 প্রস্তুত করে সর্বোচ্চ সঠিক সংখ্যা x (10 ভিত্তিতে), অনুরূপভাবে x%100 প্রস্তুত করে শেষ দুটি সংখ্যা।

৪.২ নিয়মাবলী বাস্তবায়ন:

কার্যকরী প্রোগ্রাম লিখার জন্য আমাদের প্রায় সব সময় অবস্থা নির্ধারণ করা দরকার হয় এবং সেই অনুযায়ী প্রোগ্রামের বৈশিষ্ট্য পরিবর্তন করতে হয়। নিয়মাবলীর তত্ত্ব আমাদের এই দক্ষতা প্রদান করে। সহজতম রূপটি হল “if” স্টেটমেন্ট:

```
if (x > 0) {
        System.out.println("x is positive");
}
```

বন্ধনীর মধ্যকার অংশটি হল শর্ত। এটা যদি সত্যি হয়, তাহলে বন্ধনীর মধ্যকার অংশটি বাস্তবায়িত হয়। আর শর্তটি যদি সত্যি না হয় তাহলে কিছু ঘটে না।

যদি শর্তটি কোন তুলনামূলক নিয়ন্ত্রক বহন করে, কখনো কখনো তাকে “র‍্যাশনাল অপারেটর” বলে:

```
x == y          // x equals y
x != y          // x is not equal to y
x > y           // x is greater than y
```

```
x < y          // x is less than y
x >= y         // x is greater than or equal to y
x <= y         // x less than or equal to y
```

যদিও এই কাজগুলো সম্ভবত তোমাদের কাছে পরিচিত হতে পারে, “java” গাণিতিক সংকেত থেকে কিছুটা ভিন্ন ভাষা ব্যবহার করে। একটি সাধারণ ভুল হল একটি “=” ব্যবহার করা যেখানে করা উচিত দুইটি “=” চিহ্ন। মনে রাখবে “=” এসাইনমেন্ট নিয়ন্ত্রক এবং দুইটি “=” চিহ্ন তুলনামূলক নিয়ন্ত্রক। এছাড়াও মনে রাখবে “=<” অথবা “=>” এরকম কোন চিহ্ন নেই। শর্ত নিয়ন্ত্রকের দুইটি দিক অনুরূপ হতে হবে। তুমি ints এর সাথে ints এবং double এর সাথে double তুলনা করতে পারবে।

== এবং != অপারেটরগুলো “string” এর সাথে কাজ করে, কিন্তু এগুলো তোমাদের আশানুরুপ কাজ করে না। “string” এর তুলনা কিভাবে করতে হয় তা আমরা ৮.১০ বিভাগে শিখব।

৪.৩ বিকল্প বাস্তবায়ন (Alternative Execution)

নিয়মাবলী বাস্তবায়নের দ্বিতীয় রূপটি হল বিকল্প বাস্তবায়ন, যেখানে দুটি সম্ভাবনা রয়েছে, শর্ত নির্ধারণ করে কোনটি বাস্তবায়িত হবে।
ভাষাটি অনেকটা এরূপ:

```
if (x%2 == 0) {
        System.out.println("x is even");
} else {
        System.out.println("x is odd");
}
```

যখন x 2 দ্বারা বিভাজনসাধ্য তখন অবশিষ্টাংশ যদি শূন্য হয় তাহলে আমরা জানব যে x জোড়, এবং এই কোডটি সেই অনুসারে আউটপুট প্রদান করবে। যদি শর্তটি মিথ্যা হয়, তাহলে দ্বিতীয় প্রিন্ট স্টেটমেন্টটি বাস্তবায়িত হবে। যেহেতু শর্তটি অবশ্যই সত্য অথবা মিথ্যা হতে হবে, তাই নিশ্চিতভাবে যে কোন একটি বাস্তবায়িত হবে।

এছাড়াও যদি তুমি মনে কর তুমি সংখ্যার জোড় না বিজোড় তা নির্ধারণ করতে চাও, তুমি এই কোডকে নিম্নলিখিত পদ্ধতিতে ব্যবহার করতে পার:

```
public static void printParity(int x) {
        if (x%2 == 0) {
                System.out.println("x is even");
        } else {
                System.out.println("x is odd");
        }
}
```

এখন তোমার একটি পদ্ধতি আছে printparity নামে যা তুমি যে পূর্ণ সংখ্যাটি প্রদান করতে চাও তার জন্য একটি যথার্থ বার্তা প্রিন্ট করবে। Main এ তুমি পদ্ধতিটিকে নিম্নরূপে আহ্বান করবে:

```
printParity(17);
```

সবসময় মনে রাখবে যখন তুমি কোন পদ্ধতিকে আহ্বান করবে তোমাকে আর্গুমেন্টের (argument) এর প্রকার উল্লেখ করতে হবে না। এই ধরণটি java নিজে থেকেই উদঘাটন করতে পারে। তোমাকে নিম্নলিখিত কিছু জিনিস লেখার ইচ্ছাকে দমন করতে হবে:

```
int number = 17;
printParity(int number);          // WRONG!!!
```

৪.৪ Chained conditionals

কখনো কখনো তুমি শর্ত সংক্রান্ত একটি সংখ্যা নির্ধারন করতে চাও এবং অনেক পদ্ধতির একটিকে নির্বাচন কর। একটি পদ্ধতি হল "if" এবং "else" এর একটি শ্রেণীমালার চেইন গঠন করা:

```
if (x > 0) {
    System.out.println("x is positive");
} else if (x < 0) {
    System.out.println("x is negative");
} else {
    System.out.println("x is zero");
}
```

তুমি যতদূর চাও চেইন ততদূর দীর্ঘায়িত করতে পার, যদিও এটা পড়তে কঠিন হতে পারে যদি অত্যাধিক বড় হয়ে যায়। এগুলো পড়তে সহজ করার একটি পদ্ধতি হল Indentation ব্যবহার করা, যেভাবে উদাহরণগুলোতে দেখানো হয়েছে। তুমি যদি স্টেটমেন্টগুলো সংরক্ষণ কর এবং squiggly-brackets রাখ, তোমার নিয়মাবলীতে ভুল করার সম্ভাবনা কম এবং এগুলো খুঁজে পাওয়ার সম্ভাবনা বেশি।

৪.৫ নেস্টেড নিয়মাবলী (Nested conditionals)

চেইনকরণের পাশাপাশি তুমি একটি নিয়মের মধ্যে আরেকটি নিয়মকে স্থান দিতে পার। আমরা পূর্বের উদাহরণগুলোকে এভাবে লিখতে পারি:

```
if (x == 0) {
    System.out.println("x is zero");
} else {
    if (x > 0) {
        System.out.println("x is positive");
    } else {
        System.out.println("x is negative");
    }
}
```

এখন একটি বহিরাগত নিয়ম আছে যার দুইটি শাখা রয়েছে। প্রথম শাখাটি একটি সাধারণ প্রিন্ট স্টেটমেন্ট ধারণ করে কিন্তু দ্বিতীয় শাখাটি আরেকটি নিয়মের স্টেটমেন্ট ধারণ করে যার নিজস্ব দুটি শাখা রয়েছে। ঐ শাখাগুলো দুটিই

প্রিন্ট স্টেটমেন্ট, কিন্তু সেগুলো নিয়মের (conditional) স্টেটমেন্ট হতে পারে।

পরিচিতি (Indentation) গঠনকে দৃশ্যায়িত করতে সাহায্য করে তথাপি অবস্থানকৃত নিয়মাবলী দ্রুত পড়া কঠিন হয়ে পরে। যখন পড়বে এগুলোকে পরিহার করবে।

আরেকভাবে, এরকম Nested Structure একটি সাধারণ বিষয় এবং এটা আমরা সামনে আরও দেখবো, সুতরাং তোমাদের এতে অভ্যস্ত হয়ে যাওয়াটাই উত্তম।

৪.৬ return statement

return statement তোমাকে শেষভাগে পৌঁছানোর আগেই প্রক্রিয়া বাস্তবায়ন শেষ করতে অনুমোদন দেয়। এটি ব্যবহার করার একটি কারণ হল, যাতে তুমি ভুল শর্ত খুঁজে পাও:

```
public static void printLogarithm(double x) {
        if (x <= 0.0) {
                System.out.println("Positive numbers only, please.");
                return;
        }
        double result = Math.log(x);
        System.out.println("The log of x is " + result);
}
```

এটি একটি পদ্ধতিকে সংজ্ঞায়িত করে যাকে printLogarithm বলে। যা দুবার (double) নামকৃত x কে পরিমাপ হিসাবে গ্রহণ করে। এটা দেখে x “0” (শূণ্য) থেকে ছোট না বড়। যেই ক্ষেত্রে এটি একটি বার্তাকে ভুল হিসাবে প্রিন্ট করে এবং “returns” ব্যবহার করে এই পদ্ধতিতে থেকে প্রস্থান করার জন্য। বাস্তবায়নের ধারাবাহিকতা দ্রুত ফিরে আসে কলার (caller) এর কাছে এবং পদ্ধতির অবশিষ্টাংশ লাইন বাস্তবায়িত হয় না। আমি একটি floating-point value ব্যবহার করেছিলাম শর্তের ডানপাশে কারণ বামপাশে floating-point value ছিল।

৪.৭ পরিবর্তন এর ধরন (Conversion Type)

তুমি ভাবতে পার কিভাবে “"The log of x is " + result , এর মত প্রকাশগুলো থেকে কিভাবে বের হওয়া যায় কারণ, operands গুলোর একটি string এবং অবিশিষ্টাংশগুলো দ্বিগুন। এক্ষেত্রে java আমাদের পক্ষ হতে চতুরতার পরিচয় দেয়, স্বয়ংক্রিয়ভাবে double কে string এ পরিবর্তন করে, string concatenation করার আগে।

তুমি যখনই দুটি এক্সপ্রেশনকে যোগ করতে চেষ্টা কর, এদের একটি string হলে java আরেকটিকে string এ পরিবর্তন করে এবং string concatenation সম্পন্ন করে। যদি তুমি একটি পূর্ণসংখ্যা এবং একটি ফ্লোটিং পয়েন্ট ভ্যালুর মধ্যে অপারেশন সম্পন্ন কর তাহলে কি হতে পারে বলে তোমার ধারণা?

৪.৮ Recursion (পুনরাবৃত্তি):

আমি আগের অধ্যায়ে উল্লেখ করেছি যে একটি পদ্ধতি অন্য পদ্ধতিকে আহ্বান করা বৈধ এবং আমরা এমন অনেক

উদাহরণ দেখেছি। আমি উল্লেখ করিনি যে, একটি পদ্ধতি যদি নিজেকেই আহ্বান করে সেটিও বৈধ। এটি যে একটি ভাল বিষয় তা তোমার কাছে পরিষ্কার না হতে পারে, তবে প্রোগ্রাম করার ক্ষেত্রে এটি একটি মজাদার জিনিস।

উদাহরণস্বরূপ এই পদ্ধতিটির দিকে দৃষ্টিপাত করা যাক:

```
public static void countdown(int n) {
        if (n == 0) {
                System.out.println("Blastoff!");
        } else {
                System.out.println(n);
                countdown(n-1);
        }
}
```

পদ্ধতিটির নাম countdown এবং এটি একটি পূর্ণসংখ্যা ব্যবহার করে পরিমাপক হিসাবে। যদি পরিমাপকটি শূণ্য (0) হয়, এটি “Blastoff” শব্দটি প্রিন্ট করে। অন্যথায় এটি সংখ্যাটি প্রিন্ট করে এবং countdown নামক পদ্ধতিকে আহ্বান করে নিজেই n-1 কে একটি আলোচ্য বিষয় (argument) হিসাবে প্রদান করে।

কি ঘটে যখন আমরা এই পদ্ধতিটিকে আমন্ত্রন করি “main” এ ঠিক এভাবে:

```
countdown(3);
```

countdown এর বাস্তবায়ন n=3 দিয়ে শুরু হয়। যেহেতু এর মান শূণ্য (0) নয়, এটিও ভ্যালুকে প্রিন্ট করে এবং এটি নিজেকে আহ্বান করে।

countdown পদ্ধতির বাস্তবায়ন n=2 দিয়ে শুরু হয়, এবং যেহেতু n এর মান শূণ্য নয়, এটি 2 মানকে প্রিন্ট করে এবং এটি নিজেকে আহ্বান করে।

countdown পদ্ধতির বাস্তবায়ন n=1 দিয়ে শুরু হয়, এবং যেহেতু এর মান শূণ্য নয়, এটি 1 মানকে প্রিন্ট করে এবং এটি নিজেকে আহ্বান করে।

countdown পদ্ধতির বাস্তবায়ন n=0 দিয়ে শুরু হয় এবং যেহেতু n এর মান শূণ্য নয় এটি “Blastoff!” শব্দটিকে প্রিন্ট করে এবং পরবর্তীতে ফিরে আসে।

যেই countdown এর n=1 রয়েছে, সেটি ফিরে আসে।
যেই countdown এর n=2 রয়েছে, সেটি ফিরে আসে।
যেই countdown এর n=3 রয়েছে, সেটি ফিরে আসে।

এবং তারপর তুমি “main” এ ফিরে আসো, তাহলে সম্পূর্ণ ফলাফলটি এরূপ:

```
3
2
```

```
1
Blastoff!
```

দ্বিতীয় উদাহরণ হিসেবে, পুনরায় “newline” এবং “threeLine” পদ্ধতিতে দৃষ্টিপাত করি:

```
public static void newLine() {
    System.out.println("");
}

public static void threeLine() {
    newLine(); newLine(); newLine();
}
```

যদিও এটি কাজ করে, তবে আমরা যদি দুইটি নতুন লাইন অথবা কোন একটি সংখ্যা যেমন 106 প্রিন্ট করতে চাই সেক্ষেত্রে এটি সহায়ক হবে না। বিকল্প পদ্ধতিটি হচ্ছে:

```
public static void nLines(int n) {
    if (n > 0) {
        System.out.println("");
        nLines(n-1);
    }
}
```

এই প্রোগ্রামটি “countdown” পদ্ধতির অনুরূপ, যতক্ষণ n, শূণ্য (0) থেকে বৃহত্তর হবে, এটি newline প্রিন্ট করবে এবং তারপর নিজেকে আহ্বান করবে n=1 অতিরিক্ত newline প্রিন্ট করার জন্য। 1+(n-1) হল সর্বমোট যতটি newline প্রিন্ট হয় যেটি মোটামুটি n হিসেবে বহিঃপ্রকাশ করে।

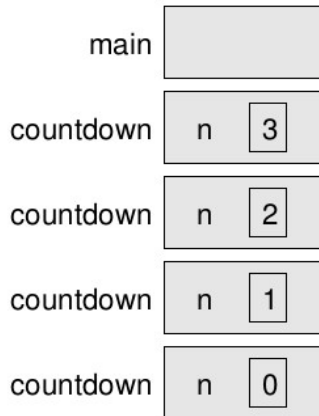
যখন একটি পদ্ধতি নিজেকে আহ্বান করে এটিকে recursion (পুনরাবৃত্তি) বলে এবং এমন পদ্ধতিকে recursive বলে।

৪.৯ পুনরাবৃত্তি (recursive) পদ্ধতির জন্য Stack diagrams

আমরা আগের অধ্যায়ে পদ্ধতি আহ্বান এর সময় প্রোগ্রামের অবস্থা উপস্থাপন করার জন্য Stack diagrams ব্যবহার করেছিলাম। একই প্রকারের ডায়াগ্রাম recursive পদ্ধতির ব্যাখ্যা সহজ করে।

মনে রাখবে যতবার একটি পদ্ধতিকে প্রয়োগ করা হয়, এটি নতুন কাঠামো তৈরি করে যার পদ্ধতি পরিমাপক এবং চলকের নতুন রূপ আছে।

নিম্নলিখিত stack diagram টি countdown পদ্ধতির যেটি n=3 এ প্রয়োগ করা হয়েছে।

main	
countdown	n 3
countdown	n 2
countdown	n 1
countdown	n 0

"main" এর জন্য একটি এবং countdown এর জন্য চারটি কাঠামো রয়েছে। যার ভিন্ন ভিন্ন মান রয়েছে n পরিমাপকের জন্য। stack diagram এর নিচে n=0 সমৃদ্ধ countdown কে base case বলে। এটি recursive কল তৈরি করেনা। সুতরাং countdown এর জন্য অতিরিক্ত কাঠামো (frame) থাকে না।

main এর জন্য কাঠামো (frame) শূণ্য (খালি) কারণ main এর কোন ক্ষেত্রে পরিমাপক অথবা চলক নেই।

৪.১০ শব্দকোষ (glossary)

মড্যুলাস: পূর্ণসংখ্যার উপর যে অপারেটর কাজ করে এবং একটি সংখ্যাকে আরেকটি সংখ্যা দ্বারা ভাগ করলে অবশিষ্টাংশ প্রস্তুত করে। java তে একে percent sign (%) দ্বারা চিহ্নিত করা হয়।

শর্তসমূহ (conditionals): একটি স্টেটমেন্ট ব্লক বাস্তবায়িত হবে কি হবে না তা নির্ভর করে কিছু শর্তের উপর।

চেইনকরণ (chaining): শর্তের স্টেটমেন্টকে শ্রেণীবদ্ধ করে যুক্ত করার পদ্ধতি।

নেস্টিং (nesting): একটি শর্তের স্টেটমেন্টকে অন্য একটি শর্তের স্টেটমেন্টে একটি অথবা দুইটি শাখায় স্থাপিত করা।

টাইপকাস্ট (type cast): একটি অপারেটর যা একটি প্রকার থেকে অন্য প্রকারে স্থানান্তর করে। java তে একটি বাক্যের সাথে সম্পর্কহীন অংশ হিসেবে আবির্ভুত হয়, যেমন- (int)।

পুনরাবৃত্তি (recursion): তোমার বর্তমান বাস্তবায়িত পদ্ধতিকে পুনরায় আহ্বান (invoking) করা।

base case: একটি অবস্থা যা পুনরাবৃত্তি (recursive) পদ্ধতিকে পুনরাবৃত্তি কল তৈরি করতে দেয়না।

৪.১১ অনুশীলনী

অনুশীলনী ৪.১: একটি stack diagram অঙ্কন কর যা ৪.৮ অধ্যায়ের প্রোগ্রামের অবস্থাকে চিত্রায়িত করে, পরিমাপক n=4 এর সহায়তায় "main" এর nLines কে আহ্বান করার পর, nLine রিটার্ন কে আহ্বান করার আগমুহূর্তে।

অনুশীলনী ৪.২: এই অনুশীলনী বিভিন্ন পদ্ধতি সমৃদ্ধ প্রোগ্রামের মধ্যদিয়ে বাস্তবায়নের ধারাবাহিকতাকে পর্যবেক্ষণ করে।

নিম্নলিখিত কোডগুলো পড় এবং প্রশ্নগুলোর উত্তর দাও:

```
public class Buzz {
    public static void baffle(String blimp) {
        System.out.println(blimp);
        zippo("ping", -5);
    }

    public static void zippo(String quince, int flag) {
        if (flag < 0) {
            System.out.println(quince + " zoop");
        } else {
            System.out.println("ik");
            baffle(quince);
            System.out.println("boo-wa-ha-ha");
        }
        }
        public static void main(String[] args) {
            zippo("rattle", 13);
        }
    }
```

১। এই প্রোগ্রামের প্রথম স্টেটমেন্ট যা বাস্তবায়িত করা হবে তারপর 1 সংখ্যাটি লিখ। যেই বিষয়গুলো স্টেটমেন্ট এবং যেগুলো নয় তাদের মধ্যে পৃথকীকরণ করতে সতর্ক থাকবে।

২। দ্বিতীয় স্টেটমেন্টটির পর 2 সংখ্যাটি লিখ এবং এভাবে প্রোগ্রাম শেষ না হওয়া পর্যন্ত। যদি স্টেটমেন্টটি একবারের বেশি বাস্তবায়িত করা হয় এটি একাধিক সংখ্যা দিয়ে শেষ হতে পারে।

৩। blimp পরিমাপকের মান কত যখন “baffle ” কে আহ্বান করা হয়?

৪। এই প্রোগ্রামের ফলাফল কি?

অনুশীলনী ৪.৩: “99 Bottles of Beer” গানটির প্রথম পংক্তি হল:

> 99 bottles of beer on the wall, 99 bottles of beer, ya’ take one
> down, ya’ pass it around, 98 bottles of beer on the wall.

পরবর্তী পংক্তিগুলো অনুরূপ, শুধু bottle সংখ্যা প্রতি পংক্তিতে কমতে থাকে, শেষ পংক্তির আগ পর্যন্ত:

> No bottles of beer on the wall, no bottles of beer, ya’ can’t take

one down, ya’ can’t pass it around, ’cause there are no more
bottles of beer on the wall!

এবং এভাবে এই গানটি শেষ হয়।

একটি প্রোগ্রাম লিখ যা, “99 Bottles of Beer” গানটি সম্পূর্ণ পংক্তিগুলো প্রিন্ট করে। তোমার প্রোগ্রামে অবশ্যই পুনরাবৃত্তিমান (recursive) পদ্ধতি থাকতে হবে যা কঠিন অংশটি করে, কিন্তু তুমি অতিরিক্ত পদ্ধতি লিখতে চাইতে পারো, প্রোগ্রামটির প্রধান কার্যগুলো আলাদা করার জন্য।

যখন তুমি তোমার কোডটি তৈরি করবে, অল্প সংখ্যাক কিছু পংক্তি দিয়ে এটিকে পরীক্ষা কর, যেমন, “o bottle of Beer” .

এই অনুশীলনীর উদ্দেশ্য হল একটি সমস্যা নির্বাচন করা এবং এটিকে ক্ষুদ্রতর সমস্যায় ভাঙ্গা এবং সাধারণ পদ্ধতি লিখার মাধ্যমে এগুলো সমাধান করা।

অনুশীলনী ৪.৪: এই প্রোগ্রামের ফলাফল কি?

```
public class Narf {

    public static void zoop(String fred, int bob) {
        System.out.println(fred);
        if (bob == 5) {
            ping("not ");
        } else {
            System.out.println("!");
        }
    }

    public static void main(String[] args) {
তত্ত্ব        int bizz = 5;
        int buzz = 2;
        zoop("just for", bizz);
        clink(2*buzz);
    }

    public static void clink(int fork) {
        System.out.print("It's ");
        zoop("breakfast ", fork) ;
    }

    public static void ping(String strangStrung) {
        System.out.println("any " + strangStrung + "more ");
    }
```

```
}
```

অনুশীলনী ৪.৫: Fermat's এর শেষ স্টেটমেন্ট বলে a, b, c কোন পূর্ণসংখ্যা নয় যদি,

$$a^n + b^n = c^n$$

শুধু মাত্র n=2 এর ক্ষেত্র ছাড়া।

check Fermat নামক একটি পদ্ধতি লিখ যা পরিমাপক হিসেবে ৪টি পূর্ণসংখ্যা নেয়। পূর্ণসংখ্যাগুলো হল, a, b, c এবং n এবং যা Fermat এর স্টেটমেন্টের সত্যতা পরীক্ষা করে। যদি n, 2 থেকে বৃহত্তর হয়, $a^n + b^n = c^n$ সমীকরণটি সত্য হয়। প্রোগ্রামটির "Holy smokes, Fermat was wrong!" প্রিন্ট করা উচিত, অন্যথায় প্রোগ্রামটির No, that doesn't work." প্রিন্ট করা উচিত।

তোমার ধারণা করা উচিত যে, raiseToPow নামক একটি পদ্ধতি রয়েছে যা দুইটি পূর্ণসংখ্যাকে আলোচ্য বিষয় হিসেবে বিবেচনা করে এবং প্রথম আলোচ্য বিষয়কে দ্বিতীয়টির শক্তি হিসেবে উত্তোলন করে। উদাহরণস্বরূপ:

```
int x = raiseToPow(2, 3);
```

8 মানটি প্রদান করবে x এর জন্য, কারণ $2^3=8$.

অধ্যায় ৫

গ্রিডওয়ার্ল্ড: পর্ব ১

৫.১ শুরু করা যাক

কম্পিউটার বিজ্ঞানের উচ্চ পর্যায়ের বিষয়গুলো নিয়ে কাজ করার এখনই উত্তম সময়। আমরা একটি প্রোগ্রাম নিয়ে কাজ করব যার নাম হচ্ছে গ্রিডওয়ার্ল্ড (GridWorld)। শুরু করার জন্য প্রথমে তোমরা গ্রিডওয়ার্ল্ড ডাউনলোড করে ইনস্টল করে নাও। এটা পাওয়া যাবে এখানে: http://www.collegeboard.com/student/testing/ap/compsci_a/case.html

যখন তোমরা এই কোড খুলবে, তখন একটি ফোল্ডার দেখতে পাবে যার নাম GridWorldCode যার ভেতর রয়েছে projects/firstProject, এবং এর ভেতর রয়েছে BugRunner.java.

BugRunner.java এর একটি অনুলিপি কর এবং অন্য একটি ফোল্ডারে তা প্রতিলেপন কর। তারপর এটিকে তোমার ডেভেলপমেন্ট এনভায়রনমেন্টের সাহায্যে খোল। এসকল বিষয়ের জন্য কিছু নির্দেশিকা তোমরা এখানে পাবে যা তোমাকে সাহায্য করবে:
http://www.collegeboard.com/prod_downloads/student/testing/ap/compsci_a/ap07_gridworld_installation_guide.pdf

BugRunner.java চালু করার পর, গ্রিডওয়ার্ল্ড স্টুডেন্ট ম্যানুয়াল ডাউনলোড কর এখান থেকে: http://www.collegeboard.com/prod_downloads/student/testing/ap/compsci_a/ap07_gridworld_studmanual_appends_v3.pdf

স্টুডেন্ট ম্যানুয়ালটি একটি শব্দতালিকা ব্যবহার করে যা আমি উপস্থাপন করিনি, তাই শুরু করার আগে, একটি তড়িৎ প্রাকদর্শন দিয়ে দিচ্ছি:

- বাগ, রকস এবং গ্রিড সহকারে গ্রিডওয়ার্ল্ডের উপাদান সমূহ হচ্ছে **অবজেক্ট (objects)**।
- **কনস্ট্রাকটর (constructor)** হচ্ছে একটি বিশেষ পদ্ধতি যা নতুন অবজেক্ট তৈরি করে।
- একটি **ক্লাশ (class)** হচ্ছে কতগুলো অবজেক্টের একটি সেট; প্রতিটি অবজেক্ট একটি ক্লাশের অংশ।
- একটি অবজেক্টকে **ইনসট্যান্স (instance)** বলা হয়ে থাকে কারন এটি একটি সদস্য, অথবা ক্লাস এর ইনসট্যান্স।
- **অ্যাট্রিবিউট (attribute)** হচ্ছে একটি অবজেক্ট সম্পর্কিত তথ্য, যেমন এর রঙ অথবা অবস্থান।
- **অ্যাকসেসর মেথড (accessor method)** হচ্ছে এমন একটি পদ্ধতি যা একটি অবজেক্টের অ্যাট্রিবিউট রিটার্ন করে।
- **মডিফায়ার মেথড (modifier method)** একটি অবজেক্টের অ্যাট্রিবিউট পরিবর্তন করে।

এখন তুমি স্টুডেন্ট ম্যানুয়াল এর এক নাম্বার পর্ব পড়তে সক্ষম হবে এবং সেই সাথে অনুশীলন করতে পারবে।

৫.২ বাগরানার (BugRunner)

BugRunner.java এর কোড টি এরকম:

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.actor.Bug;
import info.gridworld.actor.Rock;

public class BugRunner
{
        public static void main(String[] args)
        {

                ActorWorld world = new ActorWorld();
                world.add(new Bug());
                world.add(new Rock());
                world.show();
        }
}
```

প্রথম তিন লাইন হচ্ছে ইমপোর্ট স্টেটমেন্ট; এরা এই প্রোগ্রামে ব্যবহৃত গ্রিড ওয়ার্ল্ড হতে ক্লাস সমূহ তালিকাবদ্ধ করে। তুমি এই ক্লাস সমূহের ডকুমেন্টেশন পাবে এখানে:
http://www.greenteapress.com/thinkapjava/javadoc/gridworld/

অন্যান্য প্রোগ্রামের মত আমরা দেখেছি, BugRunner একটি ক্লাস নির্ধারণ করে যা একটি main মেথড প্রদান করে। main এর প্রথম লাইন একটি ActorWorld object.new তৈরি করে যেটা জাভার একটি কি ওয়ার্ড এবং এটি নতুন

অবজেক্ট তৈরি করে।

পরবর্তী দুই লাইন একটি বাগ এবং রক তৈরি করে, এবং ওয়ার্ল্ড এর সাথে যুক্ত করে। সর্বশেষ লাইন স্ক্রীনে ওয়ার্ল্ড প্রদর্শন করে।

BugRunner.java খুলে এই লাইনটি সম্পাদন এবং পরিবর্তন কর:

```
world.add(new Bug());
```

উপরের লাইনটি পরিবর্তন করে লিখ:

```
Bug redBug = new Bug();
world.add(redBug);
```

প্রথম লাইনটি Bug কে একটি ভ্যারিয়েবলে নিযুক্ত করে যার নাম redBug; Bug এর মেথড গুলো আমরা চাইলে redBug দ্বারা ব্যবহার করতে পারি। এটা করে দেখ:

```
System.out.println(redBug.getLocation());
```

টীকা: যদি তুমি এই কাজটি world এ Bug যোগ না করে চালনা কর, তাহলে ফলাফল হবে null, যার মানে হচ্ছে Bug এখনও কোন location পায় নি।

অন্যান্য অ্যাকসেসর এর মেথড bug এর অ্যাট্রিবিউট দ্বারা ব্যবহার কর। canMove, move এবং turn মেথড সমূহ ব্যবহার কর এবং তারা কি করে তা বুঝতে পারছ কি না নিশ্চিত হও।

৫.৩ অনুশীলনী

অনুশীলনী ৫.১:

১। একটি মেথড লিখ যার নাম হবে moveBug যা bug কে প্যারামিটার হিসেবে গ্রহন করে এবং move কে আহ্বান করবে। main থেকে আহ্বান করার মাধ্যমে এই মেথড পরীক্ষা কর।

২। moveBug পরিবর্তন কর যাতে এটি canMove কে আহ্বান করতে পারে এবং যদি সম্ভব হয় bug কে নাড়াতে পারে।

৩। moveBug পরিবর্তন কর যাতে এটি একটি integer, n গ্রহন করতে পারে, প্যারামিটার হিসেবে। এবং বাগ কে n সংখ্যক বার নাড়াবে (যদি সম্ভব হয়)।

৪। moveBug এমন ভাবে পরিবর্তন কর যাতে, bug নড়তে সক্ষম না হলে, এটি পরিবর্তে turn কে আহ্বান করবে।

অনুশীলনী ৫.২:

১। Math ক্লাস একটি ক্লাস প্রদান করে যার নাম random, এটি 0.0 এবং 1.0 এর মাঝে একটি double রিটার্ন করে (1.0 ব্যাতীত)।

২। randomBug নামে একটি মেথড লিখ যা Bug কে একটি প্যারামিটার হিসেবে ধরবে এবং Bug এর দিক নির্ধারণ করবে 0, 90, 180 অথবা 270 সমান সম্ভাব্যতা অনুযায়ী, এবং তারপর bug নড়বে যদি সম্ভব হয়।

৩। integer n গ্রহণ করার জন্য randomBug পরিবর্তন কর এবং n সংখ্যকবার পুনরাবৃত্তি কর। ফলাফল হবে একটি random walk, যা সম্পর্কে তুমি পড়তে পার এখানে:
http://en.wikipedia.org/wiki/Random_walk.

৪। একটি দীর্ঘ random walk দেখার জন্য, তুমি ActorWorld কে একটি বড় স্টেজ দিতে পার। BugRunner.javar এর শুরুতে, এই ইমপোর্ট স্টেটমেন্ট যোগ কর:

```
import info.gridworld.grid.UnboundedGrid;
```

এখন ActorWorld তৈরি করে যে লাইনটি তাকে পরিবর্তন কর এই লাইন দ্বারা:

```
ActorWorld world = new ActorWorld(new UnboundedGrid());
```

তুমি কয়েক হাজার স্টেপ এর জন্য তোমার random walk চালনা করতে সক্ষম হবে (Bug খুঁজে পেতে তোমাকে স্ক্রলবারে সাহায্য নিতে হতে পারে)।

অনুশীলনী ৫.৩. GridWorld ব্যবহার করে রঙের অবজেক্ট, যা জাভার লাইব্রেরীতে নির্ধারণ করে দেয়া। তুমি এই ব্যাপারে জানতে পারবে এখান থেকে: http://download.oracle.com/javase/6/docs/api/java/awt/Color.html ভিন্ন রঙে Bug তৈরি করতে চাইলে, আমাদের রঙ ইমপোর্ট করতে হবে:

```
import java.awt.Color;
```

তারপর তুমি পূর্বনির্ধারিত রঙ ব্যবহার করতে পারবে, যেমন Color.blue, অথবা এইরকম আরও নতুন রঙ:

```
Color purple = new Color(148, 0, 211);
```

ভিন্ন ভিন্ন রঙের কিছু বাগ তৈরি কর। তারপর একটি মেথড লিখ যার নাম হবে colorBug যা বাগকে প্যারামিটার হিসেবে গ্রহন করবে, এর অবস্থান জানবে, এবং রঙ নির্ধারন করবে।

getLocation হতে যে অবস্থান অবজেক্ট পেয়েছ তার মেথড হচ্ছে getRow এবং getCol যা integer রিটার্ন করে। সুতরাং তুমি বাগের x-coordinate পেতে পার যা হবে এরকম:

```
int x = bug.getLocation().getCol();
```

makeBug নামক একটি মেথড লিখ যা ActorWorld এবং একটি integer n গ্রহন করে এবং n সংখ্যক রঙিন বাগ তাদের অবস্থান অনুযায়ী তৈরি করে। লাল রঙের মাত্রা নিয়ন্ত্রণ করতে Row এর সংখ্যা এবং নীল রঙের জন্য column এর সংখ্যা ব্যবহার কর।

অধ্যায় ৬

ভ্যালু মেথড (Value methods)

৬.১ রির্টান ভ্যালু (Return values)

অংকের ফাংশন, ফল প্রস্তুত করার ফাংশনের মতো কিছু method আছে যা আমরা ব্যবহার করি। নতুন value জেনারেট (generate) করার জন্য এই ধরনের method কে আমরা ব্যবহার করি। উদাহরণস্বরূপ আমরা লিখতে পারি:

```
double e = Math.exp(1.0);
double height = radius * Math.sin(angle);
```

কিন্তু যেহেতু আমাদের method হলো void; তাই আমরা এখানে কোন value রিটার্ন পাবো না। যখন আমরা void নিয়ে কাজ করি তখন সাধারণভাবেই আমরা এখানে কোন অ্যাসাইনমেন্ট ছাড়া একটি লাইন পাই।

```
countdown(3);
nLines(3);
```

এই অধ্যায়ে আমরা এমন method নিয়ে কাজ করবো যা আমাদের কিছু রিটার্ন করবে। এগুলোকে আমরা বলছি value methods। প্রথম উদাহরণ হিসেবে আমরা area বা ক্ষেত্রকে নিতে পারি; যার parameter টাইপ হবে double এবং যা বৃত্তের প্রদত্ত ব্যাসার্ধ বা radius অনুসারে ঐ বৃত্তের ক্ষেত্রেফল বের করতে পারবে।

```
public static double area(double radius) {
        double area = Math.PI * radius * radius;
        return area;
}
```

প্রথমেই যে বিষয়টি আমাদের লক্ষ করতে হবে তা হলো, এখানে method এর শুরুটাই হয়েছে একটু অন্যভাবে। public static void, যা void method নির্দেশ করে তার পরিবর্তে আমরা ব্যবহার করেছি public static double। যার অর্থ হলো এই method এর রিটার্ন value এর ধরন বা টাইপ হবে double। কোন অর্থ পাওয়া যাচ্ছে না, তাই না? কারণ আমরা এখনো public static বলতে কি বোঝায় তাই-ই আলোচনা করিনি। সুতরাং আমাদের ধৈর্য্য ধরে অপেক্ষা করতে হবে।

উপরের উদাহরণের শেষ লাইনেও আমরা নতুন এক ধরনের রূপ দেখালাম, যা দেখালো value রিটার্ন করা। এই স্টেটমেন্ট নির্দেশ করে “method অনুসারে প্রাপ্ত ফল তাৎক্ষণিকভাবে প্রকাশ করতে এবং এই প্রকাশ হবে return value তে যেভাবে বলা হয়েছে সেভাবে।” যে expression আমরা এখানে লিখেছি তার জটিলতা এড়াতে, নিচের মতো করে আমরা আরও সংক্ষেপে লিখতে পারি।

```
public static double area(double radius) {
        return Math.PI * radius * radius;
}
```

অন্যদিকে অস্থায়ী বা temporary variable যেমন area; অনেক ক্ষেত্রে ডিবাগিং অনেক সহজ করে দেয়। রিটার্ন স্টেটমেন্ট এর expression কে অবশ্যই method এর return type-এর মতো হতে হবে। অন্যভাবে আমরা বলতে পারি, যদি আমরা কোন return type এর double হিসেবে declare করি তাহলে আমাদের নিশ্চিত করতে হবে এই method যেন ফল হিসেবে double ই প্রদর্শন করে। যদি আমরা চেষ্টা করি কোন expression ছাড়া value রিটার্ন করতে তাহলে তা হবে ভুল expression, তখন কম্পাইলার এই কাজের জন্য আমাদের কাছে জবাব চাইবে।

মাঝে মাঝে আমাদের অনেকগুলো রিটার্ন স্টেটমেন্ট ব্যবহার করতে হয় যার প্রত্যেকটিই কোন না কোন শর্তাধীন:

```
public static double absoluteValue(double x) {
        if (x < 0) {
                return -x;
        } else {
                return x;
        }
}
```

যেহেতু এখানে বিকল্প শর্ত আছে, তাই যে কোন একটি রিটার্ন স্টেটমেন্ট বাস্তবায়িত হবে। যদিও একটি method এ একাধিক রিটার্ন স্টেটমেন্ট থাকতে পারে তবুও আমাদের মনে রাখতে হবে এক সময়ে শুধুমাত্র একটি স্টেটমেন্টই বাস্তবায়িত হতে পারবে। একটি রিটার্ন স্টেটমেন্ট বাস্তবায়িত হবার পর এরপর ধারাবাহিক স্টেটমেন্ট আর বাস্তবায়িত হবে না।

রিটার্ন স্টেটমেন্ট এর পর যে কোড থাকে বা অন্য যে কোন জায়গাতে থাকা কোড যা কখনো executed হয় না তাদেরকে dead code বা মৃত কোড বলে। কিছু কিছু কম্পাইলার আছে যা আমাদের dead code এর ব্যাপারে সতর্ক করে।

যদি আমরা কোন শর্তের ভেতরে রিটার্ন স্টেটমেন্টটি বর্ণনা করি তাহলে আমাদের নিশ্চিত করতে হবে ঐ প্রোগ্রামের মধ্যে যতগুলো সম্ভাব্য পথ আছে তা ঐ রিটার্ন স্টেটমেন্টকে হিট (hit) করতে পারবে। উদাহরণস্বরূপ বলা যায়:

```
public static double absoluteValue(double x) {
        if (x < 0) {
                return -x;
        } else if (x > 0) {
                return x;
        }                                               // WRONG!!
}
```

এই প্রোগ্রামটি ভুল কারণ যদি x = 0 হয়; তাহলে এই method টি রিটার্ন স্টেটমেন্টে হিট না করেই শেষ হয়ে যাবে। আর আমরা একটি সাধারণ বার্তা পাবো কম্পাইলারের কাছ থেকে তা হচ্ছে, "রিটার্ন স্টেটমেন্ট এর প্রয়োজন absoluteValue", যার অর্থ ঠিক আমাদের কাছে পরিষ্কার নয় কারণ আমাদের কাছে রিটার্ন স্টেটমেন্টের দুটি মানই রয়েছে।

৬.২ প্রোগ্রাম তৈরি (Program development)

এখন আমরা একটি সম্পূর্ণ জাভা method দেখবো এবং বোঝার চেষ্টা করবো সেটা কীভাবে কাজ করে। কিন্তু সম্ভবত আমাদের কাছে কীভাবে সেটা লিখতে হবে তা পরিষ্কার নয়। এখন আমরা এমন একটা method নিব যার নাম **incremental development.**

উদাহরণ হিসেবে কল্পনা করি, আমরা দুটি বিন্দুর মধ্যবর্তী দূরত্ব (distance) বের করতে চাই, যাদের কো-অরর্ডিনেট বা দ্বিমাত্রিক অবস্থান হলো (x_1, x_2) এবং (x_2, y_2)। সাধারণ সূত্রানুসারে;

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

প্রথম ধাপে আমাদের চিন্তা করতে হবে জাভা কীভাবে আমাদের এই distance বের করার সমস্যাটিকে দেখবে। অন্যভাবে বললে আমাদের বলতে হবে, এক্ষেত্রে ইনপুটগুলো (parameters) কি কি এবং আউটপুটগুলো (return value) কি কি।

এক্ষেত্রে, parameters হলো দুটি পয়েন্ট বা বিন্দু আর স্বাভাবিক নিয়মে এদেরকে উপস্থাপন করা যাবে চারটি doubles parameter ব্যবহার করে। যদিও আমরা পরে দেখবো জাভাতে পয়েন্ট অবজেক্ট(Point object) আছে যা আমরা ব্যবহার করতে পারি। এখানে রিটার্ন value হলো distance, যার টাইপ double।

তাহলে এখন আমরা এই method এর একটি আউটলাইন তৈরি করি:

```
public static double distance
            (double x1, double y1, double x2, double y2) {
    return 0.0;
}
```

এই স্টেটমেন্টটি রিটার্ন করবে 0.0; এটি মূলত একটু বিরতি যা আমরা কম্পাইল করলে পাবো। মূলত এই অবস্থানে প্রোগ্রামের কোন বিশেষ অর্থ থাকে না। কিন্তু এই অবস্থানে প্রোগ্রামটি কম্পাইল করা গুরুত্বপূর্ণ কারণ এর ফলে আমরা নতুন কোড লেখার আগে বুঝতে পারি বা খুঁজে বের করতে পারি আমাদের প্রোগ্রামের কোন সিনট্যাক্স এ ত্রুটি আছে কিনা (syntax errors)।

নতুন method পরীক্ষা করার জন্য আমাদের খুব সাধারণ values নিয়ে কাজ করতে হয়। এখানে আমরা যোগ করতে পারি:

```
double dist = distance(1.0, 2.0, 4.0, 6.0);
```

আমরা এই value গুলো নিলাম যেন উলম্ব (horizontal) দূরত্ব হয় ৩ এবং আনুভূমিক (vertical) দূরত্ব হয় ৪। তাহলে আমাদের ফলাফল হবে ৫ (ত্রিভূজের ৩-৪-৫ উপপাদ্য অনুসারে)। যখন আমরা কোন method পরীক্ষা করবো তখন আমাদের জানা উচিত সঠিক উত্তরটি কি।

যখন আমরা কোন method এর সিনট্যাক্স(syntax) পরীক্ষা করবো, তখন আমরা একই সময়ে অনেক কোড সংযুক্ত করতে থাকি। প্রতিটি ক্রমবর্ধমান পরিবর্তনের পর আমাদের রিকম্পাইল করা উচিত এবং রান করে দেখা

উচিত। যদি কোন পয়েন্টে কোন ভুল থেকে থাকে তাহলে আমরা তা সহজেই খুঁজে বের করতে পারবো। সেক্ষেত্রে ভুলগুলো বেশির ভাগ ক্ষেত্রে পাওয়া যাবে নতুন সংযুক্ত করা লাইনে।

আমাদের পরবর্তী ধাপ হলো ($x_2 - x_1$) এবং ($y_2 - y_1$) এর মধ্যকার দূরত্ব বের করা। আমরা এই value গুলোকে dx এবং dy নামে দুটি অস্থায়ী variable-এর মধ্যে রাখবো।

```
public static double distance
            (double x1, double y1, double x2, double y2) {
    double dx = x2 - x1;
    double dy = y2 - y1;
    System.out.println("dx is  " + dx);
    System.out.println("dy is  " + dy);
    return 0.0;
}
```

আমরা এখানে প্রিন্ট স্টেটমেন্ট যোগ করেছি যেন প্রক্রিয়াকরণের আগে মধ্যবর্তী value সংযুক্ত করতে পারি। যা হবে 3.0 এবং 4.0।

যখন method শেষ হবে তখন আমরা প্রিন্ট স্টেটমেন্ট মুছে ফেলবো। এই ধরনের কোড লেখাকে বলে scaffolding, কারণ এই ধরনের কোডগুলো শুধু প্রয়োজন হয় প্রোগ্রাম তৈরি করার জন্য, চূড়ান্ত প্রোগ্রামের অংশ নয় এই কোডগুলো।

আমাদের এর পরের ধাপ হলো dx এবং dy এর square বের করা। আমরা এটি করার জন্য জাভার Math.pow method ব্যবহার করতে পারি আবার সাধারণভাবে প্রত্যেকটি সংখ্যাকে সেই সংখ্যা দিয়ে গুণও করতে পারি।

```
public static double distance
            (double x1, double y1, double x2, double y2) {
    double dx = x2 - x1;
    double dy = y2 - y1;
    double dsquared = dx*dx + dy*dy;
    System.out.println("dsquared is " + dsquared);
    return 0.0;
}
```

এই ধাপে পুনরায় আমি প্রোগ্রামটি কম্পাইল এবং রান করবো এবং মধ্যবর্তী মান চেক করবো। এক্ষেত্রে আমাদের মধ্যবর্তী মান হবে ২৫.০।

শেষে আমরা Math.sqrt ফাংশন ব্যবহার করবো এবং ফল রিটার্ন করবো।

```
public static double distance
            (double x1, double y1, double x2, double y2) {
    double dx = x2 - x1;
    double dy = y2 - y1;
```

```
        double dsquared = dx*dx + dy*dy;
        double result = Math.sqrt(dsquared);
        return result;
    }
```

অর্থাৎ, আমরা আমাদের উদ্দেশ্য অনুসারে ফল প্রিন্ট করতে পারলাম এবং এর value চেক বা পরীক্ষা করতে পারলাম।

যেহেতু তোমরা আরও বড় প্রোগ্রামার হবে এবং প্রোগ্রামিং আরও অনেক অভিজ্ঞতা লাভ করবে তাই হয়তো তোমাদের অনেক লাইন কোড একসাথে লিখতে এবং ডিবাগ (debug) করতে হবে। এমন ক্ষেত্রে ক্রমানুসারে প্রোগ্রামের কোড লিখে পরীক্ষা করাটা আমাদের অনেক সময় বাঁচাবে। এই প্রক্রিয়ার মূল লক্ষ্যণীয় দিকগুলো হলো:

- একটি প্রোগ্রাম নিয়ে কাজ শুরুর প্রথমে আমরা একে ছোট করে তৈরি করবো, ক্রমানুসারে এর পরিবর্তন করবো। তাহলে যদি কোন পয়েন্টে ভুল থাকে তা আমরা সহজে জানতে পারবো; ঠিক কোথায় ভুল রয়েছে।

- আমাদের অস্থায়ী ভ্যারিয়েবেল ব্যবহার করতে হবে তাহলে আমরা মধ্যবর্তী values গুলো এই অস্থায়ী ভ্যারিয়েবলে রেখে তা চেক করতে এবং পরীক্ষা করতে পারবো।

- যখন প্রোগ্রামটি কাজ করবে, তখন আমরা scaffolding সরাতে পারবো এবং অনেকগুলো স্টেটমেন্টকে একসাথে করে compound expressions ও লিখতে পারবো। কিন্তু এক্ষেত্রে আমাদের মনে রাখতে হবে সেটা যেন আমাদের প্রোগ্রামকে জটিল করে না ফেলে।

৬.৩ কম্পোজিশন (Composition)

যখন আমরা কোন নতুন method ব্যবহার করবো, আমরা তখন একে আমাদের expression-এর একটি অংশ হিসেবে ব্যবহার করতে পারি। আবার আমরা বর্তমানে ব্যবহৃত method ব্যবহার করে নতুন method ও তৈরি করতে পারি। উদাহরণস্বরূপ মনে করি, আমাদের একজন দুটি পয়েন্ট দিলো। তার একটি হলো বৃত্তের কেন্দ্র এবং অন্যটি হলো বৃত্তের পরিধিস্থ কোন বিন্দু আর আমাদের জিজ্ঞেস করা হলো বৃত্তের ব্যাসার্ধ কত?

মনে করি, কেন্দ্র বিন্দুটি দুটি xc এবং yc নামক দুটি ভ্যারিয়েবলে সংরক্ষিত বা stored করা হয়েছে এবং পরিধিস্থ পয়েন্ট দুটি হচ্ছে xp এবং yp। প্রথম ধাপে আমাদের বৃত্তের ব্যাসার্ধ বের করতে হবে, যা হলো দুটি বিন্দুর বা পয়েন্টের দূরত্ব। সৌভাগ্যবশতঃ আমাদের distance নামক একটি method আছে, যা আমাদের জন্য এই কাজ করবে।

```
double radius = distance(xc, yc, xp, yp);
```

দ্বিতীয় ধাপ হলো বৃত্তের ক্ষেত্রেফল বের করা এবং এই ফলকে রিটার্ন করা।

```
double area = area(radius);
return area;
```

এই সব method কে একসাথে যুক্ত করে আমরা পাই,

```
public static double circleArea
            (double xc, double yc, double xp, double yp) {
    double radius = distance(xc, yc, xp, yp);
    double area = area(radius);
    return area;
}
```

অস্থায়ী ভ্যারিয়েবল radius এবং area প্রোগ্রামটি তৈরি এবং ডিবাগিং-এর জন্য খুব প্রয়োজনীয়। কিন্তু যখন আমাদের প্রোগ্রামটি রান করবে তখন আমরা একে ছোট করে ফেলতে পারি, নিচের মতো করে:

```
public static double circleArea
            (double xc, double yc, double xp, double yp) {
    return area(distance(xc, yc, xp, yp));
}
```

৬.৪ ওভারলোডিং (Overloading)

আমরা হয়তো লক্ষ করে থাকবো উপরে উদাহরণে circleArea এবং area দুটিই একই রকম কাজ করছে। অর্থাৎ দুটিই বৃত্তের ক্ষেত্রফল বের করেছে, কিন্তু সেটি করেছে দুটি ভিন্ন প্যারামিটারে ব্যবহার করে। area এর জন্য আমাদের radius বা ব্যাসার্ধ এর মান লিখতে হয়েছে আবার circleArea এর জন্য আমাদের দুটি পয়েন্ট এর মান লিখতে হয়েছে।

যদি দুটি method একই কাজ করে, তাহলে খুব স্বাভাবিকভাবেই; একই নামে দুটি method না রেখে তাদের একই নাম দেওয়া যায় একেই বলে "**over-loading**"। জাভাতে এই নিয়মটি প্রতিষ্ঠিত, প্রত্যেক version বা সংস্করণই বিভিন্ন প্যারামিটার (parameters) গ্রহণ করতে পারে। তাই আমরা circleArea এর নাম পরিবর্তন করতে পারি:

```
public static double area
            (double x1, double y1, double x2, double y2) {
    return area(distance(xc, yc, xp, yp));
}
```

যখন আমরা overloaded method কে ব্যবহার করবো, জাভা জানে আমরা আমাদের আর্গুমেন্টে (argument) কোন সংস্করণ ব্যবহার করতে চাই। যদি আমরা লিখি:

```
double x = area(3.0);
```

তাহলে জাভা এমন একটি method খুঁজবে যার নাম area এবং এটি আর্গুমেন্ট হিসেবে একটি double নিয়ে কাজ করবে অর্থাৎ এটি প্রথম সংস্করণ, যা আর্গুমেন্টকে ব্যাসার্ধ হিসেবে নিয়ে নেবে। যদি আমরা লিখি:

```
double x = area(1.0, 2.0, 4.0, 6.0);
```

জাভা area এর জন্য দ্বিতীয় সংস্করণ ব্যবহার করবে। এবং আমাদের মনে রাখতে হবে area এর দ্বিতীয় সংস্করণ আসলে প্রথম সংস্করণকেই ডাকবে বা ব্যবহার করবে।

জাভার অনেক method ই overloaded। অর্থাৎ এখানে বিভিন্ন ভার্সন রয়েছে যা বিভিন্ন ধরনের নম্বর বা বিভিন্ন ধরনের প্যারামিটার (parameters) গ্রহণ করে। উদাহরণস্বরূপ বলা যায়, print এবং println এর মধ্যকার সংস্করণ একক প্যারামিটার গ্রহণ করে। আবার Math class এর ক্ষেত্রে, abs সংস্করণ যা doubles নিয়ে কাজ করে যেমন রয়েছে; তেমনি রয়েছে ints এর জন্য আর একটি সংস্করণ।

যদিও overloading একটি প্রয়োজনীয় বৈশিষ্ট্য, এটি সতর্কতার সাথে ব্যবহার করা উচিত। কখন যদি আমরা একটি সংস্করণের কোন method কে debug করতে গিয়ে দুর্ঘটনাবশত অন্য method কে invoke করি তাহলে আমরা নিজেরাই বিভ্রান্ত হয়ে যাবো।

আর এই বিষয়টিই আমাদের debugging এর মৌলিক নিয়মের কথা মনে করিয়ে দেয়: **নিশ্চিত করুন যে প্রোগ্রাম এর যে সংস্করণ আপনি খুঁজছেন আর যে প্রোগ্রাম এর যে সংস্করণ চলছে তা একই!!!**

কোন দিন হয়তো আমরা আমাদের প্রোগ্রামে একটার পর একটা পরিবর্তন করতে চাইবো, এবং দেখা যাবে প্রত্যেকবার আমরা রান করতে গিয়ে একই জিনিস খুঁজে পাবো। এটা আমাদের জন্য একটি সতর্ক বার্তা, আমাদের সবসময় সংস্করণ পরীক্ষা করে প্রোগ্রাম চালানো উচিত। এটি পরীক্ষা করার জন্য আমরা print statement ব্যবহার করতে পারি (কি প্রিন্ট করছি তা এখানে মুখ্য নয়) এবং আমাদের নিশ্চিত করতে হবে প্রোগ্রামের আচরণ আমাদের প্রত্যাশা অনুসারে পরিবর্তিত হচ্ছে।

৬.৫ বুলিয়ান এক্সপ্রেশন (Boolean expressions)

বেশির ভাগ অপারেশনের ক্ষেত্রেই আমরা দেখি, operands অনুসারেই একই রকম ফলাফল উৎপন্ন করে। উদাহরণস্বরূপ, "+" অপারেটর দুটি ints নিয়ে একটি int করে অথবা দুটি doubles নিয়ে একটি double উৎপন্ন করে।

যে ব্যতিক্রম (exceptions) আমরা দেখলাম সেগুলো relational operators, যা ints এবং floats এর মধ্যে তুলনা করে এবং true অথবা false রিটার্ন করে। জাভাতে true অথবা false এর বিশেষ values আছে এবং একসাথে তারা একটি ধরন তৈরি করে যাকে বলে **বুলিয়ান (boolean)**। আমাদের নিশ্চয়ই মনে আছে আমরা যখন কোন type নির্দিষ্ট করি তখন বলেছিলাম এর রয়েছে সুনির্দিষ্ট এক সেট values। ints, doubles এবং Strings এর ক্ষেত্রে এই সেটগুলো হয় একটু বড়। আর বুলিয়ানের জন্য রয়েছে শুধুমাত্র দুটি values।

বুলিয়ান এক্সপ্রেশন এবং ভ্যারিয়েবল অন্যান্য এক্সপ্রেশন এবং ভ্যারিয়েবল এর মতোই কাজ করে:

```
boolean flag;
flag = true;
boolean testResult = false;
```

প্রথম উদাহরণ হলো খুব সাধারণ ভ্যারিয়েবল declaration; দ্বিতীয় উদাহরণ হলো একটি assignment এবং

তৃতীয় উদাহরণ হলো initialization এর উদাহরণ।

জাভাতে true এবং false এর values হলো কিওয়ার্ড, তাই এগুলো আমাদের প্রোগ্রাম তৈরির পরিবেশ অর্থাৎ development environment বা IDE ভেদে বিভিন্ন রং এ দেখা যেতে পারে।

Conditional operator এর ফল হলো বুলিয়ান, তাই আমরা এই comparison এর ফল ভ্যারিয়েবলে সংরক্ষণ করে রাখতে পারি:

```
boolean evenFlag = (n%2 == 0);          // true if n is even
boolean positiveFlag = (x > 0);         // true if x is positive
```

এবং পরে conditional statement-এর অংশ হিসেবে একে ব্যবহারও করতে পারি:

```
if (evenFlag) {
    System.out.println("n was even when I checked it");
}
```

একটি ভ্যারিয়েবলকে এভাবে ব্যবহার করার নিয়মকে বলে flag কারণ এটি কিছু condition বা শর্তের উপস্থিতি বা অনুপস্থিতিকে চিহ্নিত করে।

৬.৬ লজিক্যাল অপারেটর (Logical operators)

জাভাতে তিনটি **Logical operators** রয়েছে: AND, OR এবং NOT, যেগুলো নির্দেশিত হয় &&, || এবং ! এই চিহ্নগুলো দ্বারা। ইংরেজিতে এই অপারেটরগুলোর শাব্দিক অর্থ যা এরা অপারেটর হিসেবে সেই কাজই করে থাকে। উদাহরণস্বরূপ বলা যায়; x > 0 && x < 10 এটি সত্যি হইবে শুধুমাত্র যদি x, 0-এর চেয়ে বড় হয় এবং (AND) 10-এর চেয়ে ছোট হয়।

evenFlag || n%3 == 0 বুঝাচ্ছে: এটি সত্যি হবে যদি এই conditions দুটির কোন একটি সত্যি হয়। অর্থাৎ evenFlag সত্যি হবে অথবা (OR) সংখ্যাটি 3 দ্বারা বিভাজ্য হবে।

Nested conditional statements কে সাধারণীকরণ করতে পারে Logical operators। উদাহরণস্বরূপ, আমরা কি এই কোডগুলোকে পুনরায় লিখতে পারবো শুধুমাত্র একটি condition ব্যবহার করে?

```
if (x > 0) {
    if (x < 10) {
        System.out.println("x is a positive single digit.");
    }
}
```

৬.৭ বুলিয়ান মেথড (Boolean methods)

অন্য যে কোন type এর মতো Methods, boolean values রিটার্ন করতে পারে। যা অনেক সময় methods এর ভেতর tests লুকানোর জন্য সুবিধাজনক। উদাহরণস্বরূপ:

```
public static boolean isSingleDigit(int x) {
        if (x >= 0 && x < 10) {
                return true;
        } else {
                return false;
        }
}
```

এই method এর নাম isSingleDigit। সাধারণত বুলিয়ান method এর নাম দেওয়া হয় এমনভাবে যেন তা হ্যাঁ/না প্রশ্নের উত্তর হিসেবে প্রদান করা যায়। এখানে রিটার্ন টাইপ হলো বুলিয়ান অর্থাৎ প্রত্যেকটি রিটার্ন statement ই বুলিয়ান এক্সপ্রেশন প্রকাশ করবে।
এক্ষেত্রে কোডগুলোও সোজা-সাপটা, কোডগুলো ঠিক ততটাই লম্বা যতটা হওয়া প্রয়োজন। আমাদের মনে আছে x >= 0 && x < 10 -এই এক্সপ্রেশনটির টাইপ হলো বুলিয়ান। তাই এখানে যদি আমরা সরাসরি একে রিটার্ন করি এবং একই সাথে if statement-কে উপেক্ষা করি:

```
public static boolean isSingleDigit(int x) {
        return (x >= 0 && x < 10);
}
```

এর অর্থ হলো আমরা এই method-কে সাধারণভাবেই ডাকতে পারি:

```
boolean bigFlag = !isSingleDigit(17);
System.out.println(isSingleDigit(2));
```

প্রথম লাইনের bigFlag সত্যি হবে শুধুমাত্র যদি 17 এক ডিজিট নম্বর না হয়।
দ্বিতীয় লাইন true প্রিন্ট করবে কারণ 2, হলো এক ডিজিট নম্বর।

Conditional statements এর ভেতর বুলিয়ান methods ব্যবহারের খুব পরিচিত উপায় হলো:

```
if (isSingleDigit(x)) {
        System.out.println("x is little");
} else {
        System.out.println("x is big");
}
```

৬.৮ আরও কিছু সূত্র (More recursion)

এখন আমাদের কাছে রয়েছে methods যা values রিটার্ন করবে। আমরা এখন প্রোগ্রামিং ল্যাঙ্গুয়েজ (programming language) এর **Turing complete** করেছি। এর অর্থ যেকোন যুক্তিযুক্ত সংজ্ঞার প্রেক্ষিতে আমরা এখন গণনা যোগ্য যেকোন কিছুই গণনা করতে পারবো। এই ধারণার জনক হলেন Alonzo Church এবং Alan Turing, তাই একে বলা হয় "Church-Turing thesis"। আমরা এ সম্পর্কে আরও জানতে পারবো http://en.wikipedia.org/wiki/Turing_thesis এই লিংক থেকে।

আমরা একটা ধারণা পেতে পারি যে tools গুলো আমরা শিখলাম তা দিয়ে আমরা কি কি করতে পারি। চলো আমরা কিছু methods দেখার চেষ্টা করি যা recursively-defined গাণিতিক ফাংশন (functions) মূল্যায়নের জন্য ব্যবহার করা হয়। Recursive এর সংজ্ঞা অনেকটা circular এর সংজ্ঞার মতোই। এই অর্থে যে সংজ্ঞা বিষয়টি হচ্ছে একটি রেফারেন্সের মাধ্যমে সংজ্ঞায়িত করার উপায়। একটি সত্যিকারের circular definition খুব প্রয়োজনীয় নয়।

recursive: recursive method বর্ণনা করার একটি বিশেষণ।

যদি আমরা শব্দকোষে এর সংজ্ঞা খুঁজতে যাই তাহলে আমরা অবশ্যই উদ্বিগ্ন হব। আবার অন্যদিকে, আমরা যদি গাণিতিক ফাংশন factorial-এর সংজ্ঞা খুঁজি তাহলে আমরা পাব:

$$0! = 1$$
$$n! = n.(n \quad 1)!$$

Factorial সাধারণত "!" এই চিহ্ন দিয়ে প্রকাশ করা হয় যা logical operator এর চিহ্নকে "! " কে বিভ্রান্ত করে না। যে logical operator বলতে আমরা বুঝি NOT। উপরের এই সংজ্ঞা অনুসারে 0 এর factorial হলো 1, এবং অন্য যে কোন value, n এর factorial হলো n এর সাথে (n-1) এর factorial এর গুণফল। অর্থাৎ, n.(n   1)!। তাহলে 3! হলো 3 এর সাথে 2! এর গুণ, আর 2! হলো 2 বার 1 কে গুণ, যার ফল 1 বার 0!। সবগুলো একসাথে লিখলে আমরা পাবো, 3! সমান 3 বার 2 বার 1 বার 1, যার ফল হলো 6।

যদি আমরা কোন কিছুর recursive definition লিখি, তাহলে খুব সহজেই আমরা একটি জাভা method লিখতে পারি একে মূল্যায়ন করার জন্য। প্রথম ধাপে আমাদের প্যারামিটারগুলো কি হবে এবং রিটার্ন টাইপ কি হবে তা ঠিক করতে হবে। যেহেতু factorial কাজ করে integers নিয়ে তাই এক্ষেত্রে method প্যারামিটার হিসেবে integer নেবে এবং রিটার্নও করবে integer।

```
public static int factorial( int n) {
}
```

If the argument happens to be zero, return 1:

```
public static int factorial( int n) {
    if (n == 0) {
        return 1;
    }
}
```

এটাই হলো মূল case।

আর এটাই হলো সবচেয়ে মজার অংশ। আমাদের recursive call তৈরি করতে হবে (n-1)-এর factorial বের করার জন্য। তারপর একে n দিয়ে গুণ করতে হবে।

```
public static int factorial(int n) {
        if (n == 0) {
                return 1;
        }       else {
                int recurse = factorial(n-1);
                int result = n * recurse;
                return result;
        }
}
```

যদি আমরা value 3 কে factorial করার জন্য invoke করি তাহলে তা সেকশন ৪.৮ এর কাউন্ট ডাউনের মতোই হলো প্রোগ্রাম execution এর flow।

যেহেতু 3, 0 (শূন্য) নয়, আমরা দ্বিতীয় ব্রাঞ্চ নিতে পারি এবং factorial (n-1) এর হিসেব করতে পারি...

যেহেতু 2, 0 (শূন্য) নয়, আমরা দ্বিতীয় ব্রাঞ্চ নিতে পারি এবং factorial (n-1) এর হিসেব করতে পারি...

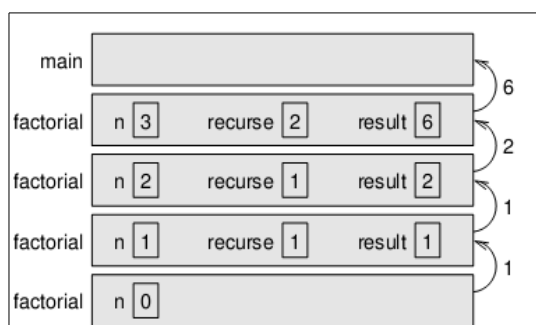
যেহেতু 1, 0 (শূন্য) নয়, আমরা দ্বিতীয় ব্রাঞ্চ নিতে পারি এবং factorial (n-1) এর হিসেব করতে পারি...

যেহেতু 0, 0 (শূন্য) নয়, আমরা দ্বিতীয় ব্রাঞ্চ নিতে পারি এবং factorial (n-1) এর হিসেব করতে পারি...

রিটার্ন value (1)-কে n দিয়ে গুণ করতে হবে, যা হবে 1 এবং এরপর ফল রিটার্ন করতে হবে।

রিটার্ন value (1)-কে n দিয়ে গুণ করতে হবে, যা হবে 2 এবং এরপর ফল রিটার্ন করতে হবে।

রিটার্ন value (2)-কে n দিয়ে গুণ করতে হবে, যা হবে 3 এবং এরপর ফল হবে 6। এই ফল মূল প্রোগ্রামে রিটার্ন হবে অথবা factorial-কে ডাকতে হবে।

এখানে method এর invocation ক্রম (sequence) এই stack diagram এ আমরা দেখতে পাবো।

এক্ষেত্রে রিটার্ন value গুলো stack এর পাশে পেছনে দেখা যাবে।

লক্ষ করি, শেষ ফ্রেমে recurse এবং result -এর কোন অস্তিত্ব নেই কারণ যখন n=0 হবে তখন যে ব্রাঞ্চ তৈরি হবে তা আর থাকবে না।

৬.৯ প্রমাণ ছাড়াই বিশ্বাস (Leap of faith)

আলোচিত flow of execution হলো যে কোন প্রোগ্রাম পড়ার একমুখী উপায় কিন্তু এটি খুব সহজেই অসামঞ্জস্য হতে পারে। এর স্থলে আমরা বলতে পারি “Leap of faith”। যখন আমরা flow of execution অনুসরণ করার বিপরীতে method invocation করবো আমরা ধরে নিতে পারি method টি সঠিকভাবে কাজ করবে এবং সঠিক value রিটার্ন করবে।

এমন কি আমরা যখন জাভা method ব্যবহার করছি তখনও আমরা কিন্তু leap of faith অনুসরণ করি। যখন আমরা Math.cos System.out.println কে ব্যবহার করি তখন আমরা এইসব methods এর বাস্তবায়ন পরীক্ষা করি না। আমরা ধরেই নেই তারা কাজ করবে।

আমরা আমাদের নিজস্ব method এর ক্ষেত্রেও এই যক্তি কাজে লাগাতে পারি। উদাহরণস্বরূপ বলা যায়, সেকশন ৬.৭ এ আমরা একটি method লিখছিলাম যাকে বলা হয়েছিল isSingleDigit যা নির্ধারণ করতো কোন সংখ্যা 0 থেকে 9-এর মধ্যে কিনা। যখন আমরা নিজেরা নিজেদেরকে কোড যাচাই এবং পরীক্ষার মাধ্যমে বোঝাতে পারলাম method টি সঠিক তখন আমরা সেই method ব্যবহার করা শুরু করলাম দ্বিতীয় বার সেই কোড আবার পরীক্ষা করা ছাড়াই।

একই বিষয় সত্যি recursive program এর জন্য। যখন আমরা recursive invocation পাবো flow of execution এর পরিবর্তে তখন আমাদের নিশ্চিত করতে হবে recursive invocation কাজ করবে। এবং তারপর নিজেদের জিজ্ঞেস করতে হবে " ধরে নেই আমি (n-1)-এর factorial বের করতে পারি তাহলে কি আমি factorial of n গণনা করতে পারবো? " হ্যাঁ, n দিয়ে গুণ করেই আমরা তা করতে পারবো।

অবশ্যই এটি খুব অবাক করার বিষয় যে আমাদের method এর কোড লেখা শেষ হবার আগেই তা সঠিকভাবে কাজ করছে, এই জন্যই একে বলা হচ্ছে leap of faith!

৬.১০ আর একটি উদাহরণ (One more example)

ফিবোনাচ্চি (fibonacci) হলো গাণিতিক ফাংশনের অতি সাধারণ উদাহরণ যা নিচের definition অনুসরণ করে।

fibonacci(0) = 1
fibonacci(1) = 1
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2);

জাভাতে আমরা যদি একে রূপান্তর করি তাহলে তা দাঁড়াবে:

```
public static int fibonacci(int n) {
        if (n == 0 || n == 1) {
```

```
            return 1;
        }   else {
            return fibonacci(n-1) + fibonacci(n-2);
        }
    }
```

যদি আমরা এখানে flow of execution করার চেষ্টা করি, এমন কি যদি n এর খুব ছোট মান নিয়েও যদি আমরা flow of execution বের করার চেষ্টা করি তাহলে আমাদের মাথায় তা কুলাবে না। কিন্তু leap of faith এর অনুসারে যদি আমরা ধরে নেই দুটি recursive invocations ঠিক মত কাজ করছে তাহলে পরিষ্কারভাবে বোঝা যায় তাদের একসাথে যুক্ত করে আমরা সঠিক ফল পেতে পারি।

৬.১১ শব্দকোষ (Glossary)

return type: method declaration এর একটি অংশ যা নির্দেশ করে কোন ধরনের value method টি রিটার্ন করবে।

return value: method invocation এর জন্য যে value দেওয়া হয়।

dead code: প্রোগ্রামের একটি অংশ যা কখনো executed হয় না কারণ এটি সবসময়ই রিটার্ন স্টেটমেন্টের পরে থাকে।

scaffolding: প্রোগ্রাম তৈরির জন্য যে কোড ব্যবহার করা হয় কিন্তু যা মূল বা ফাইনাল প্রোগ্রামের বা কোডের অংশ নয়।

void: রিটার্নের একটি বিশেষ টাইপ বা ধরন যা একটি void method নির্দেশ করে এবং তা হলো এ থেকে আমরা কোন value রিটার্ন পাই না।

overloading: একই নামে একাধিক method এর ব্যবহার, যাদের রয়েছে ভিন্ন ভিন্ন প্যারামিটার। যখন আমরা কোন overloaded method, invoke করি জাভা জানে যে আর্গুমেন্ট আমরা দিচ্ছি তা কোন ভার্সনের জন্য প্রযোজ্য।

boolean: ভ্যারিয়েবলের একটি ধরন বা টাইপ যা শুধুমাত্র সত্য এবং মিথ্যা এই দুটি value ধারণ করে।

conditional operator: একটি অপারেটর যা দুটি values এর মধ্যে তুলনা করে এবং একটি boolean উৎপন্ন করে যা operands এর মধ্যকার সম্পর্ক নির্দেশ করে।

logical operator: একটি operator যা boolean values সমন্বয়ে গঠির এবং boolean values উৎপন্ন করে।

৬.১২ অনুশীলনী (Exercises)

অনুশীলনী ৬.১: isDivisible নামে একটি method লিখো যা দুটি integers, n এবং m নিয়ে কাজ করবে এবং যা রিটার্ন true করবে যদি n, m দিয়ে বিভাজ্য হয় এবং false রিটার্ন করবে যদি তা না হয় অর্থাৎ বিভাজ্য না

হয়।

অনুশীলনী ৬.২: অনেক গণনার ক্ষেত্রেই সংক্ষিপ্ত expression এর জন্য আমরা "multadd" operation ব্যবহার করি, যা তিনটি operands নিয়ে কাজ করে এবং গণনা করে a * b + c। কিছু processors আবার কখনো কখনো floating-point নম্বরের এই operation চালানোর জন্য হার্ডওয়্যার implementation এর ব্যবস্থা রাখে।

১। নতুন একটি প্রোগ্রাম তৈরি করো যাকে Multadd.java বলা হবে।

২। একটি method লিখতে হবে যার নাম multadd এবং যা তিনটি doubles, parameters হিসেবে নিয়ে কাজ করেবে এবং তাদের multadditionization এর মান returns করবে।

৩। একটি main method লিখো যা invoking এর মাধ্যমে কয়েকটি সাধারণ প্যারামিটার যেমন 1.0, 2.0, 3.0 এর মাধ্যমে multadd যাচাই করবে ।

৪। main এর মধ্যে multadd method ব্যবহার করতে হবে নিচের values এর মান বের করার জন্য:

$$\sin\frac{\pi}{4} + \frac{\cos\frac{\pi}{4}}{2}$$
$$\log 10 + \log 20$$

৫। একটি method লিখো যাকে বলা হবে yikes এবং যা একটি double কে parameter হিসেবে নেবে এবং multadd ব্যবহার করবে নিচের গণনা করার জন্য

$$xe^{-x} + \sqrt{1 - e^{-x}}$$

ইঙ্গিত: raising e এর Math method হলো Math.exp এর ঘাত বা power।

এই শেষ অংশে, আমরা এমন একটি method লেখার সুযোগ পাবো যা আমরা যে method লিখবো তাকে invoke করবে। আমরা যাই করি না কেন, দ্বিতীয় method নিয়ে কাজ শুরু করার পূর্বে প্রথম method পরীক্ষা করার এটা হলো সবচেয়ে ভালো উপায়। অন্যথায়, আমাদের হয়তো একই সাথে দুটি method কে debugging করতে হবে, যা অনেকাংশে জটিল।

এই অনুশীলনীর অন্যতম উদ্দেশ্য হলো practice pattern-matching অর্থাৎ সাধারণ বৈশিষ্ট্যের সমস্যাগুলো থেকে বিশেষ বৈশিষ্ট্যের সমস্যাগুলো চিহ্নিত করার যোগ্যতা অর্জন।

অনুশীলনী ৬.৩: তোমাকে তিনটি কাঠি (sticks) দেওয়া হলো, তুমি হয়তো এগুলোকে ত্রিভুজের মতো সাজাতে পারো অথবা নাও পারো। উদাহরণস্বরূপ, যদি একটি কাঠি 12 ইঞ্চি লম্বা এবং অন্য দুটি one ইঞ্চি লম্বা, তাহলে তুমি কোনভাবেই ছোট কাঠি দুটিকে দিয়ে মাঝের অংশটি জোড়া দিতে পারবে না। যেকোন মাপের তিনটি দৈর্ঘ্য দিয়ে ত্রিভূজ বানানো যাবে কিনা তার সহজ পরীক্ষা হলো:

"যদি তিনটি দৈর্ঘ্যের কোন একটি অন্য দুটি দৈর্ঘ্যের যোগফলের চেয়ে বড় হয় তাহলে সে তিনটি দৈর্ঘ্য দিয়ে ত্রিভূজ বানানো যাবে না, অন্যথায় যাবে।"

একটি method লিখো যার নাম isTriangle, যা তিনটি integers, arguments হিসেবে নিয়ে কাজ করবে এবং true অথবা false রিটার্ন করবে, আর এটি নির্ভর করবে আমাদেরকে প্রদত্ত দৈর্ঘ্যের কাঠি দিয়ে আমরা ত্রিভুজ তৈরি করতে পারবো কি পারো না তার ওপর।

এই অনুশীলনীর মূল লক্ষ্য হলো value method লেখার জন্য conditional statements এর ব্যবহার।

অনুশীলনী ৬.৪: নিচের প্রোগ্রামের আউটপুট কি? এই অনুশীলনীর উদ্দেশ্য হলো তোমরা logical operators বুঝতে পারো কিনা এবং value methods এর মাধ্যমে তাদের execution করতে পারো কিনা তা নিশ্চিত করা।

```
public static void main(String[] args) {
        boolean flag1 = isHoopy(202);
        boolean flag2 = isFrabjuous(202);
        System.out.println(flag1);
        System.out.println(flag2);
        if (flag1 && flag2) {
                System.out.println("ping!");
        }
        if (flag1 || flag2) {
                System.out.println("pong!");
        }
        }
        public static boolean isHoopy(int x) {
        boolean hoopyFlag;
        if (x%2 == 0) {
                hoopyFlag = true;
        } else {
                hoopyFlag = false;
        }
                return hoopyFlag;
}


public static boolean isFrabjuous(int x) {
        boolean frabjuousFlag;
        if (x > 0) {
                frabjuousFlag = true;
        } else {
                frabjuousFlag = false;
        }
        return frabjuousFlag;
```

```
}
```

অনুশীলনী ৬.৫: দুটি বিন্দু (x_1, y_1) এবং (x_2, y_2) এর মধ্যকার দূরত্ব হলো:

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

একটি method লিখো যার নাম distance এবং যা চারটি doubles অর্থাৎ x_1, y_1, x_2 এবং y_2 কে প্যারামিটার হিসেবে নিয়ে কাজ করে এবং এই বিন্দুগুলোর মধ্যকার দুরত্ব প্রিন্ট করে।

তোমাদের জানা আছে একটি method আছে যার নাম sumSquares, যা গণনা করে পারে এবং আর্গুমেন্ট (arguments) অনুসারে sum of the squares রিটার্ন করতে পারে।

```
double x = sumSquares(3.0, 4.0);
```

Value হিসেবে আমরা x এ 25.0 অ্যাসাইন (assign) করতে পারি।

এই অনুশীলনীর মূল লক্ষ্য হলো একটি নতুন method লেখা যা বর্তমানে প্রচলিত কোন method ব্যবহার করবে। তোমাদের শুধুমাত্র distance নামক একটি method ই লিখতে হবে। তোমাদেরকে sumSquares অথবা main method লিখত হবে না এবং তোমাদেরকে distance, invoke ও করতে হবে না।

অনুশীলনী ৬.৬: এই অনুশীলনীর উদ্দেশ্য হলো recursive program এর execution বোঝার জন্য stack diagram এর ব্যবহার।

```
public class Prod {
        public static void main(String[] args) {
                System.out.println(prod(1, 4));
        }
        public static int prod(int m, int n) {
                if (m == n) {
                        return n;
                } else {
                        int recurse = prod(m, n-1);
                        int result = n * recurse;
                        return result;
                }
        }
}
```

১। একটি stack diagram আঁকো যা prod শেষ হবার ঠিক আগের instance এ প্রোগ্রামের state বা অবস্থা বোঝাবে। এই প্রোগ্রামের আউটপুট কি?

২। সংক্ষেপে prod কি কি করতে পারে তা বর্ণনা করো।

৩। অস্থায়ী ভ্যারিয়েবল (temporary variables) recurse and result ব্যবহার না করে নতুন ভাবে

prod লিখো।

অনুশীলনী ৬.৭: এই অনুশীলনীর উদ্দেশ্য হলো recursive definition কে জাভা method এ রূপান্তরিত করা। Non-negative integers গুলো কে Ackerman function এ নিম্নোক্তভাবে প্রকাশ করা হয়:

$$A(m,n) = \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m, n-1)) & \text{if } m > 0 \text{ and } n > 0. \end{cases} \quad (6.1)$$

একটি method লিখতে হবে যার নাম ack এবং যা দুটি ints প্যারামিটার হিসেবে নিয়ে কাজ করবে এবং গণনার করে Ackerman function এর value রিটার্ন করবে।

main এবং রিটার্ন value প্রিন্ট invoke করার মাধ্যমে তোমার Ackerman এর বাস্তবায়ন পরীক্ষা করো।

সর্তকীকরণ: রিটার্ন value খুব তাড়াতাড়ি বড় হতে পারে। তোমাকে m এবং n এর খুব ছোট values নিয়ে কাজ করতে হবে (2 এর চেয়ে বড় নয়)।

অনুশীলনী ৬.৮:

১। একটি প্রোগ্রাম তৈরি করো যার নাম Recurse.java এবং নিচের methods টি লিখো:

```
// first: returns the first character of the given String
public static char first(String s) {
        return s.charAt(0);
}

// last: returns a new String that contains all but the
// first letter of the given String
public static String rest(String s) {
        return s.substring(1, s.length());
}

// length: returns the length of the given String
public static int length(String s) {
        return s.length();
}
```

২। main এর মধ্যে কিছু কোড লিখো যা এই methods গুলোর সবগুলোকে পরীক্ষা করবে। নিশ্চিত করো এরা কাজ করে এবং এটাও নিশ্চিত করো এরা কি কাজ করছে তা তুমি বুঝতে পারছো।

৩। printString নামে একটি method লিখো যা একটি String কে প্যারামিটার (parameter) হিসেবে নেবে এবং তা String এর অক্ষরগুলো একটি এক লাইনে দেখিয়ে প্রিন্ট করবে। এটি হবে একটি void

method।

৪। printBackward নামে একটি method লিখতে হবে যা printString যা করে তাই করবে কিন্তু String এর অক্ষরগুলো পিছন দিকে থেকে (backward) এক লাইনে একটি অক্ষর হিসেবে প্রিন্ট করবে।

৫। reverseString নামে একটি method লিখো যা একটি String প্যারামিটার হিসেবে নিয়ে কাজ করবে এবং return value হিসেবে নতুন String রিটার্ন করবে। নতুন String টি অবশ্যই প্যারামিটার হিসেবে একই অক্ষর ধারণ করবে কিন্তু তা হবে উল্টো ভাবে। উদাহরণস্বরূপ নিচের কোডগুলোর আউটপুট:

```
String backwards = reverseString("Allen Downey");
System.out.println(backwards);
```

হবে

```
yenwoD nellA
```

অনুশীলনী ৬.৯: power নামক একটি recursive method লিখো যা একটি double x এবং একটি integer n কে নিয়ে কাজ করবে এবং যা রিটার্ন করবে x^n।

ইঙ্গিত: এই operation এর recursive definition হলো $x^n = x.x^{n-1}$ অর্থাৎ যে কোন কিছু উত্থাপিত zeroth ক্ষমতা হল 1।

Optional challenge: তোমরা এই কাজটি আরও দক্ষতার সাথে করতে পারবে যদি n এর মান জোড় সংখ্যা হয়। সেক্ষেত্রে তোমাদের ব্যবহার করতে হবে $x^n = (x^{n/2})^2$।

অনুশীলনী ৬.১০: (এই অনুশীলনীটি Ableson and Sussman's এর Structure and Interpretation of Computer Programs বই এর 44 নম্বর পৃষ্ঠা ওপর ভিত্তি করে লেখা।)

নিচের এই পদ্ধতিটি Euclid's Algorithm নামে পরিচিত কারণ এটি Euclid's Elements (Book 7, ca. 300 BC) নামক বই-এ উল্লেখ করা হয়েছে। সম্ভবত এটি সবচেয়ে পুরোনো nontrivial algorithm[1]।

এই প্রক্রিয়াটি একটি পর্যবেক্ষণের ওপর ভিত্তি করে করা আর সেটি হলো যদি a কে b দিয়ে ভাগ করলে r অবশিষ্ট থাকে তাহলে a এবং b এর সাধারণ গুণিতক (common divisors) আর b এবং r এর সাধারণ গুণিতক একই হবে। ফলে আমরা নিচের সমীকরণ ব্যবহার করতে পারি

$$gcd(a, b) = gcd(b, r)$$

GCD problem এর গণনাকালীন সময়ের সফলভাবে এর সমস্যাগুলো কমানোর জন্য GCD problem এর

1algorithm এর সংজ্ঞার জন্য, সামনে এগিয়ে যাও এবং সেকশন ১১.১৩ দেখো

গণনাকে আমাদের ছোট করে নিতে হবে আর এজন্য ছোট জোড়ার integers ব্যবহার করতে হবে। উদাহরণস্বরূপ,

$$gcd(36, 20) = gcd(20, 16) = gcd(16, 4) = gcd(4, 0) = 4$$

36 এবং 20 এর GCD হলো 4। এই একই ফল আমরা দেখতে পাব যে কোন দুটি শুরুর সংখ্যার কাছ থেকে যার পুনরাবৃত্তি হ্রাস ঘটেছে ধারাবাহিকভাবে এবং এটি একটি জোড়া তৈরি করেছে যেখানে দ্বিতীয় সংখ্যা হলো 0। তাহলে GCD হবে এই জোড়ার অন্য নম্বর।

একটি মেথড লিখ যার নাম gcd, যা দুটি পূর্ণ সংখ্যার প্যারামিটার গ্রহন করে, যা দুটি সংখ্যার সাধারণ ভাজক রিটার্ন করার জন্য ইউক্লিড এর অ্যালগরিদম ব্যবহার করে।

সপ্তম অধ্যায়
Iteration and loops (পুনরাবৃত্তি এবং লুপ)

৭.১ মাল্টিপল এসাইনমেন্ট

তুমি একই ভ্যারিয়েবলের জন্য একাধিক এসাইনমেন্ট তৈরি করতে পারো; এর ফলে এটি নতুন মান দ্বারা পুরনো মান কে প্রতিস্থাপন করবে।

```
int liz = 5;
System.out.print(liz);
liz = 7;
System.out.println(liz);
```

এই প্রোগ্রামের ফলাফল হল 57, কারণ প্রথমবার আমরা print করেছিলাম liz যার ভ্যালু ছিল 5, এবং দ্বিতীয়বার তার মান ছিল 7।

এই ধরনের মাল্টিপল এসাইনমেন্টের কারণ হিসাবে আমি যা বর্ণনা করেছি তা হল, ভ্যারিয়েবল হল ভ্যালু এর ধারক। তুমি যখন একটি ভ্যারিয়েবলের ভ্যালু এসাইন করবে, তুমি তখন ধারকের কনটেন্ট পরিবর্তন করবে, নিচে যেমনটা দেখানো হয়েছে:

int liz = 5; liz [5]

liz = 7; liz [~~5~~ 7]

যখন একটি ভ্যারিয়েবলের মাল্টিপল এসাইনমেন্ট থাকে, তখন একটি এসাইনমেন্টের statement এবং equality এর statement এর মাঝে পার্থক্য করা খুবই জরুরী। কারণ জাভা এসাইনমেন্টের জন্য = সিম্বল ব্যবহার করে। এটা একটি statement যেমন a=b কে statement of equality হিসাবে ব্যাখ্যা করার জন্য অত্যন্ত লোভনীয়। এটা তাই নয়!

প্রথমত, equality হল বিনিময় যোগ্য, এবং এসাইনমেন্ট তা নয়। উদাহরণস্বরূপ, গণিতে if a = 7 then 7 = a কিন্তু জাভায় a = 7 একটি লিগ্যাল এসাইনমেন্ট স্টেটমেন্ট এবং 7 = a; তা নয়।

এছাড়াও, গণিতে, একটি statement of equality সব সময়ের জন্যই সত্যি। যদি এখন a = b হয়, তবে সবসময়ই a b এর সমান হবে। কিন্তু জাভায়, assignment statement দুইটি ভ্যারিয়েবলকে সমান করে, কিন্তু তারা সবসময় একই ভাবে থাকে না!

```
int a = 5;
int b = a;      // a and b are now equal
a = 3;          // a and b are no longer equal
```

তৃতীয় লাইন a এর ভ্যালু পরিবর্তন করে কিন্তু এটি b এর ভ্যালু পরিবর্তন করে না। তাই তারা একইভাবে সমান না। কিছু প্রোগ্রামিং ল্যাঙ্গুয়েজে এই বিভ্রান্তি দুর করতে এসাইনমেন্টের জন্য একটি পৃথক সিম্বল ব্যবহার করা হয়, যেমন <- অথবা := ।
যদিও মাল্টিপল এসাইনমেন্ট প্রায়ই ব্যবহার করা হয়, তুমি এটি সতর্কতার সাথে ব্যবহার করতে পারো। যদি ভ্যারিয়েবলের ভ্যালু প্রায়ই পরিবর্তন হয় তবে এটি কোডকে রিড এবং ডিবাগ করতে কঠিন করে তৈরি করে।

৭.২ while স্টেটমেন্ট

কম্পিউটারকে প্রায়ই স্বয়ংক্রিয়ভাবে পুনরাবৃত্তিমূলক কাজ করতে ব্যবহার করতে হয়। ভুল করা ছাড়া পুনরাবৃত্তিমূলক কাজ অনেকটা এমন যেটা কম্পিউটার খুব ভালো ভাবে করে কিন্তু মানুষ দুর্বলভাবে করে থাকে।

আমরা আগেই দেখেছি যে, countdown এবং factorial এই জাতীয় মেথডগুলো গাণিতিক ফর্মুলায় ব্যবহার করা হয় পুনরাবৃত্তিমূলক কাজ করানোর জন্য। এই প্রক্রিয়াকেই বলা হয় iteration। এই মেথডগুলো লেখার জন্য জাভাতে সহজ কিছু উপায় রয়েছে। এই অধ্যায়ে আমরা while statement এর ব্যবহার লক্ষ্য করব। পরবর্তীতে (১২.৪ অংশে) আমরা for statement পরীক্ষা করব।

while statement ব্যবহার করে আমরা countdown কে পুনরায় লিখতে পারি:

```
public static void countdown(int n) {
        while (n > 0) {
                System.out.println(n);
                n = n-1;
        }
        System.out.println("Blastoff!");
}
```

তুমি প্রায়ই ইংরেজির মতো করে while statement টি পড়তে করতে পার। এর অর্থ কি হবে, “যখন n শূণ্যের চেয়ে বড়, n এর মান প্রিন্ট কর এবং পরে n এর মান ১ করে কমাতে থাক। যখন তুমি শূণ্য পাবে, তখন ‘Blastoff!” ’ প্রিন্ট কর।

আরও আনুষ্ঠানিক করে বলতে গেলে, while statement এর কাজ করার ধারাবাহিক পদ্ধতি নিচে দেওয়া হল:

১। প্যারেনথিসিসের অবস্থা বা condition নির্ণয় কর, এটি true না false.
২। যদি কন্ডিশন false হয়, while statement বন্ধ করে দাও এবং পরবর্তী statement এর এক্সিকিউট শুরু করে দাও।
৩। যদি কন্ডিশন true হয়, তবে ব্রাকেটের ভেতরের statement গুলো এক্সিকিউট করতে হবে এর পর প্রথম ধাপে যেতে হবে।

এই রকমের ফ্লো কে বলা হয়ে থাকে loop কারণ তৃতীয় ধাপটি পুনরায় প্রথম ধাপে নিয়ে যায়। loop এর ভেতরে থাকা statement কে বলা হয় loop এর body। যদি কন্ডিশন false হয় তবে প্রথমবার loop এর কাজ করা হবে। loop এর ভেতর থাকা statement টি এক্সিকিউট করবে না।

লুপের বডি এক বা একাধিক ভ্যারিয়েবলের মানকে পরিবর্তন করে দিতে পারে তাই, পরিনামে, কন্ডিশন false হতে পারে এবং loop এর কাজ শেষ হয়ে যেতে পারে। অন্যথায় loop চিরকাল পুনরাবৃত্ত হতেই থাকবে। কম্পিউটার বিজ্ঞানীদের পর্যবেক্ষণের জন্য একটি সীমাহীন কৌতুককর সোর্স হল শ্যাম্পু এর মতো গতিপথ, " ফেনা তৈরি করা, আলতো করে ধুয়ে ফেলা, পুনরায় কাজগুলো করা," হল একটি অসীম loop।

countdown এর ক্ষেত্রে, আমরা প্রমাণ করতে পারি যে, যদি n ধনাত্মক হয় তবে loop শেষ হয়ে যাবে। অন্যক্ষেত্রে এটা কি হবে বলা কষ্ট:

```
public static void sequence(int n) {
    while (n != 1) {
        System.out.println(n);
        if (n%2 == 0) {          // n is even
            n = n / 2;
        } else {                 // n is odd
            n = n*3 + 1;
        }
    }
}
```

এই loop এর জন্য কন্ডিশন হল n !=1, তাই এই লুপটি n 1 হওয়া পর্যন্ত চলতে থাকবে, যা কন্ডিশনকে false করবে।

প্রতিটি পুনরাবৃত্তে, প্রোগ্রামটি n এর ভ্যালু প্রিন্ট করবে এবং তারপর পরীক্ষা করবে এটি জোড় না বিজোড়। যদি এটি জোড় হয়, n এর মান দুই দ্বারা ভাগ করা হবে। যদি বিজোড় হয়, তবে মানটি 3n+1 দ্বারা পরিবর্তন করা হবে। উদাহরণস্বরূপ, যদি প্রারম্ভিক মান (আর্গুমেন্ট সিকুয়েন্স হিসেবে সম্পন্ন হয়) 3 হয়, তাহলে ফলাফলের ধারাবাহিকতা হল, 3, 10, 5, 16, 8, 4, 2, 1 .

যেহেতু n কখনও বাড়ে কখনও কমে, এর কোন পরিষ্কার নিদর্শন নেই যে, n কখন 1 এ পৌঁছাবে, অথবা এই প্রোগ্রামটি শেষ হবে। n এর কিছু নির্দিষ্ট মানের জন্য আমরা প্রমাণের সমাপ্তি করতে পারি। উদাহরণস্বরূপ, যদি একটি প্রারম্ভিক মান হয় power of two, তাহলে n এর মান প্রতিবার জোড় হবে এবং লুপ হতেই থাকবে যতক্ষণ না আমরা 1 পাই। পূর্বের উদাহরণটি শেষ হয়েছে একটি ধারাবাহিকতায় যা শুরু হয়েছিল 16 থেকে।

আলাদা মানের জন্য আমরা কিভাবে বলতে পারি যে, n এর সকল মানের জন্য প্রোগ্রামটি শেষ হয়েছে। তাই, এটি কেউ প্রমাণ করতে পারে না, অথবা মিথ্যাও প্রমাণ করতে পারে না! আরও তথ্যের জন্য http://en.wikipedia.org/wiki/Collatz_conjecture এটা দেখা যেতে পারে।

৭.৩ টেবিল (Tables)

সারণীবদ্ধ ডেটা জেনারেট এবং প্রিন্ট করার জন্য loops অনেক কাজের। উদাহরণস্বরূপ, কম্পিউটারের বহুল প্রচলনের আগে, আমরা লগারিদম, সাইন, কোসাইন এবং অন্যান্য ম্যাথমেটিক্যাল ফাংশনগুলোর হিসেব হাতেই করতাম।

এই কাজকে সহজ করার জন্য, একটি বই থাকতো যাতে একটি বৃহৎ টেবিলে নানান ফাংশনের জন্য নির্ধারিত মানগুলো দেওয়া থাকতো এবং সহজে তা খুঁজে পাওয়া যেত। এই টেবিল তৈরি করা একটি দীর্ঘ সময় ও বিরক্তিকর কাজ ছিল এবং এই ফলাফল তখন ভুলে পূর্ণ থাকত।

যখন এই কাজে কম্পিউটারের প্রবেশ ঘটলো, তখন প্রারম্ভিক প্রতিক্রিয়াই ছিল, “এটা দারুণ! আমরা কম্পিউটারকে সারণী জেনারেটের কাজে ব্যবহার করতে পারি, যাতে কোন ভুল না থাকে।” এটা অনেকাংশে সত্যি কিন্তু এটারও সীমাবদ্ধতা আছে। এরপরে কম্পিউটারের ব্যবহার এতো ব্যাপক হল যে টেবিল অপ্রচলিত হয়ে পড়লো।

কিছু অপারেশনের জন্য, একটি আনুমানিক উত্তর পেতে কম্পিউটার মানের টেবিল ব্যবহার করে এবং তারপর গণনা করতে শুরু করে আনুমানিক মানটির উন্নতি করতে। কিছু ক্ষেত্রে, অন্তর্নিহিত টেবিলে কিছু ত্রুটি বিদ্যামান থাকে, সবচেয়ে বিখ্যাত টেবিল ইন্টেলের পেন্টিয়াম টেবিলটি ব্যবহার করা হয় floating-point ডিভিশনের কাজে।[2]

যদিও একটি “লগ সারণী” খুব একটা দরকারি না, তবে এটি একটি খুব ভালো পুনরাবৃত্তির উদাহরণ হতে পারে। নিচের প্রোগ্রামটি বামপাশের কলামে মানের একটি ধারাবাহিকতা প্রিন্ট করে এবং ডানপাশের কলামে তাদের লগারিদম প্রিন্ট করে।

```
double x = 1.0;
while (x < 10.0) {
        System.out.println(x + "     " + Math.log(x));
        x = x + 1.0;
}
```

এই প্রোগ্রামের ফলাফল হবে:

```
1.0    0.0
2.0    0.6931471805599453
3.0    1.0986122886681098
4.0    1.3862943611198906
5.0    1.6094379124341003
6.0    1.791759469228055
7.0    1.9459101490553132
8.0    2.0794415416798357
```

2 http://en.wikipedia.org/wiki/Pentium_FDIV_bug.

9.0 2.1972245773362196

এই মানের দিকে লক্ষ্য কর, তুমি কি বলতে পারবে, লগ পদ্ধতির কি বেস এখানে ব্যবহার করা হয়েছে?

যেহেতু দুই এর ঘাত (powers of two) কম্পিউটার বিজ্ঞানে গুরুত্বপূর্ণ, তাই আমরা প্রায়ই চাই ২ ভিত্তিক (base 2) লগারিদম। এই হিসাব করতে আমরা নিচের ফর্মুলাটি ব্যবহার করে থাকি:

$\log_2 x = \log_e x / \log_e 2$

প্রিন্ট স্ট্যাটমেন্ট পরিবর্তন করে পাই:

```
System.out.println(x + "     " + Math.log(x) / Math.log(2.0));
```

এক্ষেত্রে ফলাফল পাই

```
1.0     0.0
2.0     1.0
3.0     1.5849625007211563
4.0     2.0
5.0     2.321928094887362
6.0     2.584962500721156
7.0     2.807354922057604
8.0     3.0
9.0     3.1699250014423126
```

আমরা দেখতে পাই যে, 1, 2, 4 এবং 8 দুইয়ের ঘাত (power), কারণ তাদের লগারিদম বেইস 2 হচ্ছে একটি পূর্ণ সংখ্যা। যদি আমরা অন্যান্য দুইয়ের পাওয়ারের লগারিদম খুঁজে বের করতে চাই, তাহলে আমাদের প্রোগ্রামটিকে পরিবর্তন করতে হবে নিচের উপায়ে:

```
double x = 1.0;
while (x < 100.0) {
        System.out.println(x + "     " + Math.log(x) / Math.log(2.0));
        x = x * 2.0;
}
```

এখন x এর সাথে কোন কিছু যোগ করার পরিবর্তে প্রতিবার loop এর ভেতর দিয়ে যেতে হবে, যেটা একটি গাণিতিক ক্রম তৈরি করে, আমরা x কে কোন কিছু দিয়ে ভাগ করতে পারি যেটা, একটি জ্যামিতিক ক্রম তৈরি করে। ফলাফল হবে:

```
1.0     0.0
2.0     1.0
4.0     2.0
```

8.0 3.0
16.0 4.0
32.0 5.0
64.0 6.0

লগ সারণী কিছু সময় উপকারী নাও হতে পারে, কিন্তু কম্পিউটার বিজ্ঞানীদের জন্য, দুই এর ঘাত জানতেই হবে! যখন তুমি অলস সময় কাটাবে তখন তুমি মনে করতে চেষ্টা করবে 65536 এর দুই এর ঘাত কত (এটা হল 2^{16})।

৭.৪ দ্বি-মাত্রিক সারণী (Two-dimensional tables)

একটি দ্বিমাত্রিক সারণী কলাম এবং সারি ধারণ করে যা ছেদের মান খুব সহজে খুঁজে পেতে সহায়তা করে। গুণের সারণী এর একটি খুব ভালো উদাহরণ। ধরা যাক তুমি 1 থেকে 6 এর গুণের সারণী প্রিন্ট চাও।

এর ভালো একটি উপায় হবে একটি খুব সাধারণ loop লেখা যা দুই এর গুণন এক লাইনে প্রিন্ট করবে। নিচের উদাহরণটি দেখ:

```
int i = 1;
while (i <= 6) {
        System.out.print(2*i + "     ");
        i = i + 1;
}
System.out.println(" ");
```

প্রথম লাইনে i নামে একটি ভ্যারিয়েবলের সূচনা করে, যেটা একটি counter হিসাবে, অথবা একটি loop ভ্যারিয়েবল হিসাবে ভুমিকা পালন করে। loop এক্সিকিউটের সময় i এর মান 1 থেকে 6 পর্যন্ত বাড়তে থাকে; যখন i7 হয়, তখন loop এর কাজ শেষ হয়। loop এর মাধ্যমে প্রত্যেকবার, আমরা 2*i এর মান এবং তিনটি spaces প্রিন্ট করি। যেহেতু আমরা System.out.print ব্যবহার করি তাই ফলাফলটি একটি একক লাইনে প্রদর্শিত হয়।

কিছু ক্ষেত্রে print এর আউটপুট সংরক্ষণ করা হয়ে থাকে, যতক্ষণ পর্যন্ত না println যুক্ত করা হয়। যদি প্রোগ্রামটি বন্ধ হয়ে যায়, এবং তুমি যদি println যুক্ত করতে ভুলে যাও, তুমি কখনও সংরক্ষিত আউটপুট দেখতে পাবে না।

উপরের প্রোগ্রামের আউটপুট হবে এইরকম:

2 4 6 8 10 12

এখানে এই পর্যন্তই, পরবর্তী ধাপে encapsulate, generalize নিয়ে আলোচনা করা হবে।

৭.৫ এনক্যাপসুলেশন এবং সর্বজনীন (Encapsulation and generalization)

এনক্যাপসুলেশন বলতে বোঝায় একটি কোডের অংশ নেয়া এবং একে একটি মেথডে মোড়ানো, এটি তোমাকে অনুমতি দিবে মেথডের সবগুলো সুবিধা কাজে লাগানোর। যখন সেকশন ৪.৩ এ printParity এবং সেকশন ৬.৭ এ isSingleDigit ব্যবহার করেছিলাম তখন এই দুটি উদাহরণে আমরা এনক্যাপসুলেশন এর ব্যবহার দেখেছিলাম।

সর্বজনীন বলতে বোঝায় কোন কিছু নির্দিষ্ট করে নেওয়া, যেমন দুইয়ের গুণিতক প্রিন্ট করা, এবং এটিকে সাধারণ

করা, যেমন যেকোন integer এর গুণিতক প্রিন্ট করা।

এখানে একটি মেথড আছে যা পূর্ববর্তী সেকশনের loop কে এনক্যাপসুলেট (আবদ্ধ) করবে এবং সর্বজনীন করে n এর গুণিতক প্রিন্ট করবে।

```
public static void printMultiples(int n) {
        int i = 1;
        while (i <= 6) {
                System.out.print(n*i + "     ");
                i = i + 1;
        }
        System.out.println("");
}
```

এনক্যাপসুলেট এ, প্রথম লাইনে সব যুক্ত করতে বলা হয়েছিল, যা নাম, প্যারামিটার এবং রিটার্ন টাইপ ডিক্লেয়ার করে। সর্বজনীনে, ভ্যালু 2 এর সাথে প্যারামিটার n কে পরিবর্তিত করা হয়েছিল।

যদি আমরা আর্গুমেন্ট 2 এ এই মেথডটি যুক্ত করি, আমরা একই ফলাফল অর্থাৎ পূর্বের ফলাফলটি পাবো। আর্গুমেন্ট 3 এর সাথে, ফলাফলটি হবে:

```
3       6       9       12      15      18
```

এবং আর্গুমেন্ট 4 এর সাথে ফলাফলটি হবে:

```
4       8       12      16      20      24
```

এখন তুমি অনুমান করতে পারবে, কিভাবে একটি গুণন সারণী তৈরি করা হয়: আমরা বিভিন্ন আর্গুমেন্টের সাথে printMultiples যুক্ত করে এটি করতে পারি। আসলে, আমরা অন্য একটি loop সারির ভেতর বারে বারে ব্যবহার করতে পারি।

```
int i = 1;
while (i <= 6) {
        printMultiples(i);
        i = i + 1;
}
```

প্রথমত, লক্ষ্য কর, কিভাবে এই loop একটি printMultiples এর ভেতরে একই হয়। এখানে আমি সবগুলো প্রিন্টকে invocation মেথড দিয়ে প্রতিস্থাপন করেছি।

এই প্রোগ্রামের আউটপুট হবে:

```
1    2    3    4    5    6
2    4    6    8    10   12
3    6    9    12   15   18
4    8    12   16   20   24
5    10   15   20   25   30
6    12   18   24   30   36
```

যেটা একটি (সামান্য এলোমেলো) গুণিতক সারণী। যদি এলোমেলোটি তোমার জন্য বিরক্তিকর হয় তবে, জাভা কিছু মেথড প্রোভাইড করে থাকে যা দিয়ে তুমি ফলাফলের ফরমেট নিয়ন্ত্রণ করতে পারবে, কিন্তু আমি এখানে এটি দিতে পারছিনা।

৭.৬ মেথড এবং এনক্যাপসুলেশন (Methods and encapsulation)

মেথডগুলো কেন দরকারী তার কিছু কারণ আমি সেকশন ৩.৫ এ লিপিবদ্ধ করেছিলাম। এখানে আরও কিছু দেওয়া হল:

- স্ট্যাটমেন্টের ধারাবাহিকতার নাম করণ করা হলে, তুমি তোমার প্রোগ্রামকে সহজেই রিড এবং ডিবাগ করাতে পারবে।

- মেথডের মাঝে একটি বড় প্রোগ্রামকে খণ্ডে খণ্ডে ভাগ করতে পারো, আলাদা করে ডিবাগ করতে পারো, এবং তারপর তুমি একটি সমগ্র ভাবে এটি কম্পোজ করতে পারো।

- মেথডগুলো রিকার্শন এবং ইটারেশনকে সহজতর করে।

- সুগঠিত মেথডগুলো অনেক প্রোগ্রামের জন্যই প্রয়োজনীয়। একবার একটি লিখে ডিবাগ করে পুনরায় তুমি এটি ব্যবহার করতে পারো।

এনক্যাপসুলেশন এর ব্যবহার পুনরায় দেখাতে, আমি পূর্বের সেকশন থেকে কিছু কোড নিয়ে তাকে একটি মেথডে মুড়িয়ে দিলাম:

```
public static void printMultTable() {
        int i = 1;
        while (i <= 6) {
                printMultiples(i);
                i = i + 1;
        }
}
```

যে ডেভেলপমেন্ট প্রসেসটি আমি প্রদর্শন করেছি তাকে বলা হয় এনক্যাপসুলেশন এবং জেনারেলাইজেশন। তুমি main বা অন্য মেথডে কোড যোগ করা শুরু করবে পারবে। যখন তুমি দেখবে যে এটি কাজ করছে, তুমি এটিকে সম্প্রসারণ করতে পারবে এবং তুমি একটি মেথডে এটি মোড়াতে পারবে। এরপর তুমি এতে প্যারামিটার যুক্ত করে মেথডের সর্বজনীন করতে পারবে।

মাঝে মাঝে তুমি জানো না ঠিক কখন তুমি প্রোগ্রামকে কিভাবে মেথডে ভাগ করবে। এই প্রক্রিয়াটি তোমাকে একা

একা ডিজাইন করতে শেখাবে।

৭.৭ লোকাল ভ্যারিয়েবল (Local variables)

তুমি হয়তো আশ্চর্য হবে যে, কিভাবে আমরা একই ভ্যারিয়েবল i কে printMultiples এবং printMultTable এ ব্যবহার করেছি। আমি কি বলেছি তুমি একটি ভ্যারিয়েবল একবারই ডিক্লেয়ার করতে পারবে? এবং যখন একটি মেথডের একটি ভ্যারিয়েবলের একটি ভ্যালু পরিবর্তন হবে তখন এটি সমস্যা সৃষ্টি করবে না?

দুইটি প্রশ্নেরই উত্তর "না", কারণ printMultiples এ i এবং printMultTable এ i একই ভ্যারিয়েবল না। তাদের একই নাম, কিন্তু তারা একই সংরক্ষণের স্থান চিহ্নিত করে না এবং একটির পরিবর্তনে অপরটিতে কোন প্রভাব পড়ে না।

একটি মেথডের ভেতরে ভ্যারিয়েবল কল করা হলে তাকে local variable বলে, কারণ তারা শুধু মাত্র মেথডের ভেতর বিদ্যামান থাকে। তুমি বাইরের "home" মেথড থেকে local variable এ প্রবেশ করতে পারবে না, এবং তুমি একই নামে একাধিক ভ্যারিয়েবল ব্যবহার করতে পারবে, যতক্ষণ পর্যন্ত না তারা একই মেথডের হয়।

যদিও এটা বিভ্রান্তিকর হতে পারে, তবে পুনরায় নামগুলো ব্যবহারের অনেক ভালো কারণও আছে। উদাহরণস্বরূপ, loop ভ্যারিয়েবলের জন্য i, j এবং k হল কয়েকটি কমন নাম। তুমি এগুলো অন্য জায়গায় ব্যবহার করেছো বলে এই মেথডে ব্যবহার করতে না চাও, তাহলে তুমি তোমার প্রোগ্রামকে অনেক কঠিন করে ফেলবে।

৭.৮ আরও কিছু সর্বজনীন (generalization)

অন্য আরেকটি সর্বজনীনের এর উদাহরণ হল, কল্পনা কর তুমি একটি প্রোগ্রাম চেয়েছিলে যেটা শুধু মাত্র 6x6 নয়, যেকোন সাইজের একটি মাল্টিপ্লিকেশন টেবিল প্রিন্ট করতে পারে। তুমি printMultTable এ একটি প্যারামিটার যোগ করতে পারো:

```
public static void printMultTable(int high) {
        int i = 1;
        while (i <= high) {
                printMultiples(i);
                i = i + 1;
        }
}
```

আমি উচ্চ প্যারামিটারের সাথে 6 পরিবর্তন করেছি। যদি আমি আর্গুমেন্ট ৭ এর সাথে printMultTable যুক্ত করি তাহলে আমি পাবো:

```
1    2    3    4    5    6
2    4    6    8    10   12
3    6    9    12   15   18
4    8    12   16   20   24
5    10   15   20   25   30
6    12   18   24   30   36
```

7 14 21 28 35 42

কোনটা ভালো, কিছু ব্যতিক্রম ছাড়া আমি সম্ভবত চাইবো টেবিলটি বর্গাকৃতি হোক (সমান সংখ্যক কলাম এবং সারি), যেটার মানে হল আমি অন্য একটি প্যারামিটার যোগ printMultiples করেছি, যেখানে কতটি কলাম থাকবে তা নির্দিষ্ট করে দিয়েছি।

এছাড়াও আমি উচ্চ প্যারামিটার কল করব, পরিচিতি করাবো যে, ভিন্ন মেথডে একই নামে প্যারামিটার থাকতে পারে (local ভেরিয়েবলে যেমন হয়):

```
public static void printMultiples(int n, int high) {
        int i = 1;
        while (i <= high) {
                System.out.print(n*i + "     ");
                i = i + 1;
        }
        System.out.println("");
}

public static void printMultTable(int high) {
        int i = 1;
        while (i <= high) {
                printMultiples(i, high);
                i = i + 1;
        }
}
```

লক্ষ্য কর, যখন আমি একটি নতুন প্যারামিটার যোগ করেছিলাম, তখন প্রথম লাইন পরিবর্তন করেছিলাম, এবং যখন মেথডে printMultTable যোগ করেছিলাম তখন এর স্থানও পরিবর্তন করেছিলাম। আশা করেছিলাম এই প্রোগ্রাম 7x7 টেবিলে রূপান্তর হবে:

```
1   2   3   4   5   6   7
2   4   6   8   10  12  14
3   6   9   12  15  18  21
4   8   12  16  20  24  28
5   10  15  20  25  30  35
6   12  18  24  30  36  42
7   14  21  28  35  42  49
```

যখন তুমি একটি মেথডকে যথাযথ ভাবে সর্বজনীন করবে, তখন তুমি প্রায়ই খুঁজে পাবে যে তুমি যার পরিকল্পনা কর নাই তাও করার ক্ষমতা এর আছে। উদাহরণস্বরূপ, তুমি প্রায়ই লক্ষ্য করবে যে নামতার টেবিলটি প্রতিসম, কারণ ab=ba, তাই টেবিলে প্রতিটি মানই জোড়ায় জোড়ায় দেওয়া আছে। তুমি এই টেবিলের অর্ধেক প্রিন্ট করে কালি বাঁচাতে পারো। এই কাজটি করতে, printMultTable এর একটি লাইনে পরিবর্তন করতে হবে। পরিবর্তনটি হবে:

```
        printMultiples(i, high);
```
থেকে
```
        printMultiples(i, i);
```

এবং তুমি পাবে,

```
1
2       4
3       6       9
4       8       12      16
5       10      15      20      25
6       12      18      24      30      36
7       14      21      28      35      42      49
```

এটি কিভাবে কাজ করে তা খুঁজে বের করার দায়িত্ব তোমার।

৭.৯ শব্দকোষ (Glossary)

লুপ (loop): এটি একটি স্ট্যাটমেন্ট, যা যখন কোন কন্ডিশন সঠিকভাবে কাজ করে তখন পুনরায় কাজটি শুরু করে।

অসীম লুপ (infinite loop): একটি লুপ যার কন্ডিশন সবসময়ই সত্যি (true)।

body: loop এর ভেতর একটি স্ট্যাটমেন্ট।

iteration: body এর ভেতর দিয়ে যাওয়া loop যার সাথে কন্ডিশনের evaluation যুক্ত থাকে।

encapsulate: জটিল কোন প্রোগ্রামকে কম্পোনেন্ট এ ভাগ করে নেওয়া (যেমন মেথডগুলো) এবং কম্পোনেন্টগুলোকে একে অপরের কাছ থেকে পৃথক করা (উদাহরণস্বরূপ, local ভ্যারিয়েবল ব্যবহার করা)

local ভ্যারিয়েবল: একটি ভ্যারিয়েবল যা মেথডের ভেতর ডিক্লেয়ার করা হয় এবং যেটা শুধু মাত্র মেথডের ভেতরই বিদ্যামান থাকে। local ভ্যারিয়েবল তাদের মূল মেথডের বাইরে এক্সেস করতে পারে না, এবং অন্য মেথডেও হস্তক্ষেপ করতে পারে না।

সর্বজনীন (generalize): অনাবশ্যকভাবে সুনির্দিষ্ট করে সাধারণ কোন কিছুর পরিবর্তন করা (যেমন কনস্টেন্ট ভ্যালু)। সর্বজনীন একটি প্রোগ্রামের কোডকে নানাবিধ ব্যবহার উপযোগী, পুনরায় ব্যবহার করতে আরও আকর্ষণীয়, এবং অনেক সময় রাইট করার জন্য সহজতর করে তোলে।

প্রোগ্রামের ডেভেলপমেন্ট: প্রোগ্রাম রচনার একটি প্রক্রিয়া। পরবর্তীতে আমরা দেখবো, “incremental development” এবং “encapsulation and generalization” ।

৭.১০ উদাহরণ

উদাহরণ ৭.১: নিচের কোডটি মনোযোগ দিয়ে দেখ:

```
public static void main(String[] args) {
        loop(10);
}

public static void loop(int n) {
        int i = n;
        while (i > 0) {
                System.out.println(i);
                if (i%2 == 0) {
                        i = i/2;
                } else {
                        i = i+1;
                }
        }
}
```

১। একটি টেবিল আঁক, যেটা ভ্যারিয়েবল i এবং n এর মান loop এক্সিকিউশনের সময় দেখাবে। টেবিলটি অবশ্যই প্রতিটি ভ্যারিয়েবলের জন্য একটি কলাম এবং প্রতিটি iteration এর জন্য একটি লাইন দেখাবে।

২। এই প্রোগ্রামের ফলাফল কি হবে?

উদাহরণ ৭.২: ধরা যাক তোমাকে একটি নম্বর, a দেওয়া হল এবং বলা হল এর স্কয়ার রুট কত হবে তা খুঁজে বের কর। এটি করার একটি উপায় হল, খুব খারাপভাবে এর উত্তর সম্পর্কে অনুমান করে তুমি এটি শুরু করতে পারো। তখন তুমি এর উত্তর বলবে x_0 এবং পরবর্তীতে তুমি তোমার অনুমানকে উন্নত করবে নিচের সূত্রের সাহায্যে:

$$x_1 = (x_0 + a/x_0)/2$$

উদাহরণস্বরূপ, যদি তুমি 9 এর স্কয়ার রুট খুঁজে বের করতে চাও, এবং তুমি $x_0 = 6$ থেকে শুরু কর তাহলে পাবে $x_1 = (6 + 9/6)/2 = 15/4 = 3.75$ যেটা প্রায় নিকটবর্তী।

আমরা এই প্রক্রিয়াকে পুনরায় করতে পারি, x_1 ব্যবহার করে x_2 এবং আরও অনেক কিছু হিসেব করতে পারি। এই ক্ষেত্রে, x_2=3.075 এবং x_3=3.00091 যেটা খুব দ্রুত সঠিক উত্তরের দিকে (যেটা হল 3) এগিয়ে যাচ্ছে।

একটি মেথড squareRoot কল কর, যেটা double কে প্যারামিটার হিসাবে গ্রহণ করবে এবং প্যারামিটারের সঠিক স্কয়ার রুট রিটার্ন করবে। তুমি অবশ্য Math.sqrt ব্যবহার করতে পারবেনা।

তোমার প্রাথমিক অনুমানে, তুমি a/2 ব্যবহার করেছিলে। তোমার মেথড পুনরাবৃত্ত হবে যতক্ষণ পর্যন্ত না এটা দুইটি ধারাবাহিক হিসাব পায় যেটা 0.0001 এর কম থেকে ভিন্ন হয়; অন্যভাবে বলা যায়, যতক্ষণ পর্যন্ত না x_n-x_{n-1} এর মান 0.0001 এর কম হয়। তুমি সঠিক মান হিসাবের জন্য Math.abs ব্যবহার করতে পারো।

উদাহরণ ৭.৩: উদাহরণ ৬.৯ এ আমরা power এর রিকার্সিভ ভার্সন রাইট করেছিলাম, যেখানে একটি ডাবল x এবং একটি পূর্ণসংখ্যা n গ্রহণ করে এবং x^n রিটার্ন করে। এখন একই হিসাব করতে একটি iterative মেথড

ব্যবহার কর।

উদাহরণ ৭.৪: সেকশন ৬.৮ এ একটি রিকার্সিভ মেথড উপস্থাপন করা হয়েছে, যেটা ফ্যাকটোরিয়াল ফাংশন হিসাব করতে পারে। এখন factorial ফাংশনের একটি iterative ভার্সন রাইট কর।

উদাহরণ ৭.৫: e^x হিসাব করার একটি উপায় হল infinite সিরিজের বিস্তার

$$e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + ...$$

যদি loop ভ্যারিয়েবল এর নাম i হয়, তাহলে i তম টার্ম হবে $x^i/i!$..

1. myexp নামে একটি মেথড রাইট কর, যেটা এই সিরিজের প্রথম n কে এড করে। তুমি সেকশন ৬.৮ এ ব্যবহৃত factorial মেথড ব্যবহার করতে পারো অথবা পূর্ববর্তী উদাহরণের iterative ভার্সন ব্যবহার করতে পারো।
2. তুমি এই মেথডকে আরও বেশি কার্যকরী করতে পারো, যদি তুমি বুঝতে পারো যে, প্রতিটি পুনরাবৃত্ত মানে এর পূর্বসরী লব সমান হয় যেটা x দ্বারা গুণিতক এবং যেটার হর পূর্বের i এর গুণিতকের সমান হয়। এই পর্যবেক্ষণটি Math.pow এবং factorial বর্জনে ব্যবহার কর এবং পরীক্ষা কর যে তুমি সমান মান পেয়েছো।
3. check নামে একটি মেথড রাইট কর, যেটা একটি সিংগেল প্যারামিটার x গ্রহণ করে এবং যেটা x এর মানকে প্রিন্ট করে, Math.exp(x) এবং myexp(x) x এর বিভিন্ন মান। এর ফলাফল কিছুটা নিচের মতো হবে:

   1.0 2.708333333333333 2.718281828459045

   ইঙ্গিত: তুমি কলামের ভেতর একটি ট্যাব ক্যারেকটার প্রিন্ট করতে "nt" স্ট্রিং টি ব্যবহার করতে পারো।

4. এই সিরিজের নম্বরগুলো পরিবর্তন কর (দ্বিতীয় আর্গুমেন্ট যেটা myexp কে check প্রদান করে) এবং রেজাল্টে এর সঠিক প্রভাব দেখ। যতক্ষণ পর্যন্ত না সঠিক উত্তর (যখন x 1 হয়) সাথে না মিলে ততক্ষণ পর্যন্ত এটা ঠিক কর।

5. main এ একটি loop রাইট কর, যেটা 0.1, 1.0, 10.0, এবং 100.0 মানের সাথে check যুক্ত করে। কিভাবে x পরিবর্তনের সাথে মানের সঠিকতা পরিমাপ করবে? এগ্রিমেন্ট এর সাথের ডিজিটের নম্বরগুলোর তুলনা কর, যতক্ষণ পর্যন্ত না সেটা সঠিক এবং অনুমানকৃত নম্বরের সাথে পার্থক্য না করে।
6. main এ একটি loop যুক্ত কর, যেটা -0.1, -1.0, -10.0, এবং -100.0 মানের সাথে myexp কে পরীক্ষা করে।নির্ভুলতার উপর মন্তব্য কর।

উদাহরণ ৭.৬: $exp(-x^2)$ পরিমাপের আরেকটি উপায় হল infinite সিরিজের বিস্তার

$$exp(-x^2) = 1 - x^2 + x^4/2 - x^6/6 + . . .$$

অন্য কথায়, আমাদের একটি সিরিজ যোগ করতে হবে যেখানে i তম টার্মটি $(-1)^i x^{2i}/i!$ এর সমান। gauss নামে একটি মেথড রাইট কর, যেটা x এবং আর্গুমেন্ট হিসাবে n নেয় এবং সিরিজের প্রথম n এর যোগফলকে রিটার্ন করে। তুমি অবশ্যই factorial অথবা pow ব্যবহার করতে পারবে না।

অধ্যায় ৮

স্ট্রিং এবং অন্যান্য উপকরণ (Strings and things)

৮.১ ক্যারেক্টার

জাভা এবং অন্যান্য অবজেক্ট ওরিয়েন্টেড ভাষায়, object হচ্ছে সম্পর্কযুক্ত ডাটার সমাহার (collection of related data) যার রয়েছে একটি মেথডের সেট। এই মেথড সমূহ অবজেক্ট পরিচালনা করে, গণনা কার্য সম্পাদন করে এবং মাঝে মধ্যে অবজেক্টের ডাটা পরিবর্তন ও সম্পাদনার কাজ করে থাকে।

স্ট্রিং হচ্ছে অবজেক্ট, তুমি হয়ত প্রশ্ন করতে পার "স্ট্রিং অবজেক্টের মধ্যে সংরক্ষিত ডাটা তাহলে কি?" এবং "স্ট্রিং অবজেক্ট আহ্বান করার জন্য মেথডসমূহ কি?" স্ট্রিং অবজেক্টের উপাদানসমূহ হল বর্ণ অথবা সাধারণভাবে বলতে গেলে অক্ষর (characters)। সকল অক্ষর কিন্তু বর্ণ নয়; কিছু রয়েছে সংখ্যা, বিভিন্ন চিহ্ন, এবং অন্যান্য জিনিস। সহজবোধ্য হওয়ার জন্য আমি এদের সবকিছুকেই বর্ণ (letters) বলব। অসংখ্য মেথড বিদ্যমান, তবে আমি মাত্র কিছু মেথড এই বইয়ে ব্যবহার করেছি। বাকি সকল মেথড তোমরা পাবে এখানে:
http://docs.oracle.com/javase/6/docs/api/java/lang/String.html

প্রথম যে মেথডটি আমরা দেখব সেটি হচ্ছে charAt, যা একটি স্ট্রিং হতে বর্ণ (letters) বাছাই করতে অনুমোদন করে। char হচ্ছে এক ধরনের ভ্যারিয়েবল যা একক characters মজুদ করতে পারে।

chars কাজ করে ঠিক আগে আমরা যেভাবে দেখেছি সেভাবেই:

```
char ltr = 'c';
if (ltr == 'c') {
        System.out.println(ltr);
}
```

ক্যারেক্টার অর্থাৎ অক্ষর এর মান প্রতীয়মান হয় একক উদ্ধৃতি চিহ্ন যেমন 'c' এর মাধ্যমে। স্ট্রিং ভ্যালুর মত (যা দ্বৈত উদ্ধৃতি চিহ্ন দ্বারা লিখা হয়) ক্যারেক্টার ভ্যালু একাধিক বর্ণ অথবা চিহ্ন ধারণ করতে পারে না।

আমরা দেখি কিভাবে charAt মেথড ব্যবহার করা হয়:

```
String fruit = "banana";
char letter = fruit.charAt(1);
System.out.println(letter);
```

fruit.charAt() এর অর্থ হচ্ছে আমি fruit নামক অবজেক্টে charAt মেথড আহ্বান করছি। আমি এই মেথডে আর্গুমেন্ট 1 পাঠাচ্ছি, যার অর্থ হচ্ছে আমি স্ট্রিং এর প্রথম বর্ণ দেখতে চাই। ফলাফল হচ্ছে একটি ক্যারেক্টার, যা মজুদ থাকে char এ letter নামে। যখন আমি ফলাফল আউটপুটে দেখতে চাই, আমি বিস্মিত হই:

```
a
```

যদি তুমি একজন কম্পিউটার বিজ্ঞানী না হয়ে থাক তাহলে বলবে “banana” শব্দটির প্রথম বর্ণ a নয়। প্রযুক্তিগত কারণে কম্পিউটার বিজ্ঞানীগণ শূন্য থেকে গণনা শুরু করে। “banana” শব্দটির শূন্যতম বর্ণ হচ্ছে b। প্রথম বর্ণ হচ্ছে a দ্বিতীয় বর্ণ হচ্ছে n।
যদি তুমি স্ট্রিং এর শূন্যতম বর্ণ দেখতে চাও তাহলে আর্গুমেন্ট হিসেবে তোমাকে শূন্য পাঠাতে হবে:

```
char letter = fruit.charAt(0);
```

৮.২ দৈঘ্য (Length)

আমরা পরবর্তী যে স্ট্রিং মেথডটি দেখব সেটি হচ্ছে দৈর্ঘ্য (length), যা স্ট্রিং এ ক্যারেক্টার রিটার্ন করে। উদাহরণস্বরূপ:

```
int length = fruit.length();
```

length কোন আর্গুমেন্ট গ্রহণ না করে একটি পূর্ণসংখ্যা (integer) রিটার্ন করে, এই ক্ষেত্রে 6। লক্ষণীয় যে, মেথডের যে নাম সেই নামে একটি ভ্যারিয়েবল থাকা বৈধ (যদিও আমাদেরকে বিষয়টি দ্বিধান্বিত করতে পারে)।

স্ট্রিং এর সর্বশেষ বর্ণ খুঁজে পেতে, তুমি হয়ত নিচের মত করে চেষ্টা করবে:

```
int length = fruit.length();
char last = fruit.charAt(length);          // WRONG!!
```

এটা কাজ করবে না। কারণ হচ্ছে “banana” তে কোন ষষ্ঠ বর্ণ নেই। যেহেতু আমরা গণনা শুরু করেছি 0 থেকে, ষষ্ঠ বর্ণ সংখ্যায়িত হবে 0 থেকে 5 এর মধ্যে। সর্বশেষ ক্যারেক্টার পেতে হলে, তোমাকে দৈর্ঘ্য থেকে 1 বাদ দিতে হবে।

```
int length = fruit.length();
char last = fruit.charAt(length-1);
```

৮.৩ ট্র্যাভেরসাল (Traversal)

প্রাথমিক পর্যায়ে একটি স্ট্রিং এর সাথে করার মত সাধারণ বিষয় হচ্ছে, প্রতি ক্যারেক্টার নির্বাচন করা, তাদের গণনা কার্য সম্পাদন করা, এবং শেষ না হওয়া পর্যন্ত চালিয়ে যাওয়া। এই ধরনের কাজকে বলা হয় ট্র্যাভেরসাল। স্বাভাবিক উপায়ে একটি ট্র্যাভেরসাল এনকোড করার উপায় হচ্ছে While স্টেটমেন্ট ব্যবহার করা:

```
int index = 0;
while (index < fruit.length()) {
    char letter = fruit.charAt(index);
    System.out.println(letter);
    index = index + 1;
}
```

এই লুপ টি স্ট্রিং ট্র্যাভার্স করে এবং নিজে থেকেই লাইন অনুসারে প্রিন্ট হয়। লক্ষ্য কর যে, কন্ডিশন হচ্ছে index < fruit.length() এর মানে হচ্ছে যখন index স্ট্রিং এর দৈর্ঘ্যের সমান, তখন কন্ডিশনটি মিথ্যা হয় এবং লুপ টি কাজ করে না। সর্বশেষ যে ক্যারেক্টার আমরা একসেস করব তা হল index fruit.length()-1

লুপ ভ্যারিয়েবলের নাম হচ্ছে index. একটি index হচ্ছে ভ্যারিয়েবল অথবা ভ্যালু যা নির্দিষ্ট সেট এর সদস্য চিহ্নিত করতে ব্যবহৃত হয়, এই ক্ষেত্রে ক্যারেক্টার এর স্ট্রিং। এই index নির্দেশিত করে (অর্থাৎ নাম) তোমার যেটা প্রয়োজন সেটা।

৮.৪ রান-টাইম ত্রুটি (Run-time errors)

সেকশন ১.৩.২ তে আমরা রান-টাইম ত্রুটি নিয়ে কথা বলেছি, এরা হচ্ছে এমন ধরনের ত্রুটি যা প্রোগ্রাম আরম্ভ হওয়ার আগ পর্যন্ত প্রদর্শিত হয় না। জাভাতে রান-টাইম ত্রুটিকে বলা হয় এক্সসেপশন **(exceptions)**।

তুমি সম্ভবত খুব বেশী রান-টাইম ত্রুটি দেখ নি, কারণ আমরা এমন কিছুই করিনি যার ফলে এটা তৈরি হবে। ভাল কথা, এখন আমরা সেটা করে দেখব। যদি তুমি charAt মেথড ব্যবহার করে থাক এবং একটি index প্রদান করে থাক যা ঋণাত্মক অথবা length-1 এর থেকে বৃহৎ, এটি তখন এক্সসেপশন throw করবে।

এটি যখন ঘটবে, জাভা একটি ত্রুটি বার্তা প্রদর্শন করবে যেখানে থাকবে এক্সসেপশন এর ধরন এবং একটি stack trace, যা তোমাকে জানিয়ে দিবে কোন মেথড চলাকালীন এই ত্রুটি ঘটেছে। একটি উদাহরণ দেখ:

```
public class BadString {
    public static void main(String[] args) {
        processWord("banana");
    }
    public static void processWord(String s) {
        char c = getLastLetter(s);
        System.out.println(c);
    }

    public static char getLastLetter(String s) {
        int index = s.length();                 // WRONG!
        char c = s.charAt(index);
        return c;
    }
}
```

getLastLetter এ ত্রুটি লক্ষ্য কর: the index of the last character should be s.length()-1 . তুমি যে ত্রুটি বার্তা দেখবে:

Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String index out of range: 6

```
at java.lang.String.charAt(String.java:694)
at BadString.getLastLetter(BadString.java:24)
at BadString.processWord(BadString.java:18)
at BadString.main(BadString.java:14)
```

তারপর প্রোগ্রামটি শেষ হয়। stack trace পড়া কঠিন মনে হতে পারে, কিন্তু এতে অনেক তথ্য থাকে।

৮.৫ ডকুমেন্টেশন পড়া (Reading documentation)

তুমি যদি http://docs.oracle.com/javase/6/docs/api/java/lang/String.html এই লিঙ্কে যাও এবং charAt এ ক্লিক কর, তুমি নিচের মত একটি ডকুমেন্টেশন পাবে:

public char charAt (int index)

Returns the char value at the specified index. An index ranges from 0 to length() - 1. The first char value of the sequence is at index 0, the next at index 1, and so on, as for array indexing.

Parameters: index - the index of the char value.

Returns: the char value at the specified index of this string. The first char value is at index 0.

Throws: IndexOutOfBoundsException - if the index argument is negative or not less than the length of this string.

প্রথম লাইনটি হচ্ছে মেথডের প্রোটোটাইপ, যা নির্দিষ্ট করে মেথডের নাম বলে দেয়, প্যারামিটার এর ধরন এবং রিটার্ন টাইপ বলে দেয়।

পরবর্তী লাইন মেথডের কাজ ব্যাখ্যা করে। তার পরের লাইন প্যারামিটার এবং রিটার্ন ভ্যালু বর্ণনা করে। এই ক্ষেত্রে ব্যাখ্যা প্রয়োজনাতিরিক্ত, কিন্তু ডকুমেন্টেশন এর আদর্শ ফরম্যাটও রক্ষা করতে হবে। সর্বশেষ লাইন ব্যাখ্যা করে এটি কী ধরনের এক্সসেপশন throw করতে পারে।

এই ধরনের ডকুমেন্টেশন এর সাথে অভ্যস্থ হতে কিছুটা সময় লাগতে পারে, কিন্তু এই প্রচেষ্টা মূল্যহীন নয়।

৮.৬ indexOf মেথড (The indexOf method)

charAt এর উল্টোটি হচ্ছে indexOf। charAt একটি index গ্রহণ করে সেই index এ character রিটার্ন করে; indexOf একটি ক্যারেক্টার গ্রহণ করে এবং এই ক্যারেক্টার যেখানে প্রদর্শিত হয় সেটার index খুঁজে বের করে।

index যদি শেষ হয়ে যায় তাহলে charAt ব্যর্থ হয়, এবং একটি এক্সসেপশন throw করে। indexOf ব্যর্থ হয় যদি স্ট্রিং এ ক্যারেক্টার প্রদর্শিত না হয় এবং এটার ফলে -1 ভ্যালু রিটার্ন করে।

```
String fruit = "banana";
int index = fruit.indexOf('a');
```

এটি স্ট্রিং থেকে “a” বর্ণ টি খুঁজে বের করে। এই ক্ষেত্রে, বর্ণটি তিন বার প্রদর্শিত হয়, সুতরাং indexOf আসলে কি করছে তা বোধগম্য হচ্ছে না। ডকুমেন্টেশন অনুসারে, এটা প্রথম প্রদর্শিত “a” বর্ণ কে রিটার্ন করে।

পরবর্তী বর্ণগুলো খুঁজতে, indexOf এর অন্য আরেকটি সংস্করণ রয়েছে। এটা দ্বিতীয় আর্গুমেন্ট নিয়ে নেয় যাতে সে বুঝতে পারে কোথায় খুঁজতে হবে। এই জাতীয় ওভারলোডিং এর ব্যাখ্যা সম্পর্কে বিস্তারিত জানতে ৬.৪ অংশ দেখ।

আমরা যদি আহ্বান করি:

```
int index = fruit.indexOf('a', 2);
```

এটা আরম্ভ হবে দ্বিতীয় বর্ণ থেকে এবং দ্বিতীয় “a” খুঁজে নিবে, যা রয়েছে index ৩ এ। যদি এটা index এর শুরু থেকে আরম্ভ হয়, তাহলে শুরুর index টিই হবে ফলাফল। তাহলে,

```
int index = fruit.indexOf('a', 5);
```

রিটার্ন করবে ৫।

৮.৭ লুপিং এবং কাউন্টিং (Looping and Counting)

নিচের প্রোগ্রামটি একটি স্ট্রিং এ “a” বর্ণ কতবার এসেছে তা গণনা (Count) করবে:

```
String fruit = "banana";
int length = fruit.length();
int count = 0;

int index = 0;
while (index < length) {
    if (fruit.charAt(index) == 'a') {
        count = count + 1;
    }
    index = index + 1;
}
System.out.println(count);
```

এই প্রোগ্রামটি একটি সাধারণ উপভাষার প্রমাণ দেয়, এর নাম কাউন্টার **(counter)**। count ভ্যারিয়েবলটি প্রাথমিক অবস্থায় শূণ্য থাকে এবং প্রতিবার 'a' খোঁজার সময় বৃদ্ধি পেতে থাকে। **increment** হচ্ছে এক এক করে বৃদ্ধি পাওয়া; এটা **decrement** এর উল্টো। আমরা যখন লুপ থেকে বের হই, count এর ভেতর ফলাফল থেকে যায়: a এর মোট সংখ্যা।

৮.৮ বৃদ্ধি এবং হ্রাস এর অপারেটর (Increment and decrement operator)

Increment এবং decrement খুব বেশী প্রচলিত অপারেটর যে জাভা তাদের জন্য কিছু বিশেষ অপারেটর প্রদান করে। ++ অপারেটর বর্তমান ভ্যালুর int অথবা char সাথে ১ যোগ করে। -- ১ করে বাদ দেয়। এই অপারেটর এর একটিও double, booleans অথবা string এর জন্য কাজ করে না।

টেকনিক্যালি, একই সময়ে কোন ভ্যারিয়েবল এর increment করা এবং কোন এক্সপ্রেশনে তা ব্যবহার করা বৈধ। উদাহরণস্বরূপ তোমরা এটি হয়ত দেখে থাকবে:

```
System.out.println(i++);
```

এটা দেখে বোঝা যাচ্ছে না যে increment এর প্রভাব ভ্যালুটি প্রিন্ট হওয়ার পূর্বে ঘটবে নাকি পরে। এই ধরনের এক্সপ্রেশন দ্বিধার সৃষ্টি করে, সুতরাং আমি এভাবে এর ব্যবহার করাকে অনুৎসাহিত করব। প্রকৃতপক্ষে তোমাকে আরও অনুৎসাহিত করতে আমি এর ফলাফল তোমাকে জানাব না। তুমি যদি সত্যিই জানতে আগ্রহী হও তাহলে নিজে চেষ্টা কর।

increment অপারেটর ব্যবহার করে, আমরা letter-counter পুনরায় লিখতে পারি:

```
int index = 0;
while (index < length) {
        if (fruit.charAt(index) == 'a') {
                count++;
                }
                index++;
        }
```

নিচের মত করে লিখাটা খুবই সাধারণ ত্রুটি:

```
index = index++;                    //WRONG ! !
```

দুর্ভাগ্যবশত, সিনট্যাক্স অনুযায়ী এটি বৈধ, সুতরাং কম্পাইলার তোমাকে সতর্ক করবে না। এই স্টেটমেন্টের প্রভাব হচ্ছে index এর ভ্যালু অপরিবর্তনশীল রেখে দেয়া। এই বাগ সমাধান করা মাঝেমধ্যেই দুরূহ হয়ে উঠে।

মনে রাখবে তুমি index = index+1 লিখতে পার অথবা index++ লিখতে পার কিন্তু এদেরকে মিশিয়ে ফেলবে না।

৮.৯ স্ট্রিং সমূহ অপরিবর্তনীয় (Strings are immutable)

তুমি যদি স্ট্রিং মেথড এর ডকুমেন্টেশন পড়ে থাক, তাহলে হয়ত তুমি toUpperCase এবং toLowerCase লক্ষ্য করে থাকবে। এই মেথডগুলো প্রায়শই দ্বিধার উৎস, কারণ এটি শুনলে মনে হয় তারা বিদ্যমান স্ট্রিং পরিবর্তন করবে। প্রকৃতপক্ষে, এই মেথডগুলো অথবা অন্য কোন কিছুই স্ট্রিং পরিবর্তন করতে পারে না, কারণ স্ট্রিং হচ্ছে অপরিবর্তনীয় (immutable)।

যখন তুমি স্ট্রিং এ toUpperCase আহ্বান করবে, তুমি রিটার্ন ভ্যালু হিসেবে একটি নতুন স্ট্রিং পাবে।

উদাহরণস্বরূপ:

```
        String name = "Alan Turing";
        String upperName = name.toUpperCase();
```

দ্বিতীয় লাইন এক্সিকিউট হওয়ার পর, upperName ধারণ করে "ALAN TURING" ভ্যালুটি, কিন্তু নাম তখনও "Alan Turing" থেকে যায়।

৮.১০ স্ট্রিং অতুলনীয় (Strings are incomparable)

মাঝে মধ্যে একাধিক স্ট্রিং এর ভেতর তুলনা করে দেখতে হয় তারা একই কিনা, অথবা বর্ণানুক্রমিক ভাবে কোনটি আগে আসছে। এটা ভাল হত যদি আমরা তুলনা করার অপারেটর যেমন == এবং > ব্যবহার করতে পারতাম, কিন্তু আমরা সেটা পারব না।
স্ট্রিং তুলনা করার জন্য আমাদের equals এবং compareTo মেথড ব্যবহার করতে হবে।
উদাহরণস্বরূপ:

```
String name1 = "Alan Turing";
String name2 = "Ada Lovelace";

if (name1.equals (name2)) {
        System.out.println("The names are the same.");
}

int flag = name1.compareTo (name2);
if (flag == 0) {
        System.out.println("The names are the same.");
} else if (flag < 0) {
        System.out.println("name1 comes before name2.");
} else if (flag > 0) {
        System.out.println("name2 comes before name1.");
}
```

এখানের সিনট্যাক্স কিছুটা অদ্ভুত। দুটি স্ট্রিং তুলনা করার জন্য, একটির উপর তোমাকে একটি মেথড আহ্বান করতে হবে এবং অন্যটিকে আর্গুমেন্ট হিসেবে পাস করতে হবে।

equals এর রিটার্ন ভ্যালু যথেষ্ট সোজাসাপটা; যদি স্ট্রিং একই ক্যারেক্টার ধারণ করে তাহলে এটি সত্য, অন্যথায় এটি মিথ্যা।

compareTo এর রিটার্ন ভ্যালুও অদ্ভুত। স্ট্রিং এর প্রথম ক্যারেক্টারগুলোর পার্থক্যই এদেরকে আলাদা করে। যদি স্ট্রিং সমান হয়, তাহলে ০। যদি প্রথম স্ট্রিং (যেখান থেকে মেথড আহ্বান করা হয়েছে) বর্ণক্রমের প্রথমে আসে, তখন পার্থক্য হবে ঋণাত্মক। অন্যথায়, পার্থক্য হবে ধনাত্মক। এই ক্ষেত্রে রিটার্ন ভ্যালু ধনাত্মক ৮, কারণ "Ada" এর দ্বিতীয় অক্ষর "Alan" এর দ্বিতীয় অক্ষরের পর আসে।

শুধুমাত্র পূর্ণতা আনার জন্য, আমাকে বলতেই হয় স্ট্রিং এর সাথে == ব্যবহার করা বৈধ, কিন্তু সবসময় সঠিক হয় না। ১৩.4 অংশে আমি এটা নিয়ে ব্যাখ্যা কর; এখনকার জন্য এটা করার প্রয়োজন নেই।

৮.১১ শব্দকোষ (Glossary)

অবজেক্ট (Object): একটি সম্পর্কযুক্ত ডাটা'র সংগ্রহ যার অপারেট করার জন্য কতগুলো মেথডের সেট থাকে। আমরা এ পর্যন্ত যতগুলো অবজেক্ট ব্যবহার করেছি তা হচ্ছে Strings, Bugs, Rocks, এবং অন্যান্য গ্রিডওয়ার্ল্ড অবজেক্ট।

ইনডেক্স (index): কোন নির্দিষ্ট সেটের (যেমন স্ট্রিং এর ক্যারেক্টার এর সেট) সদস্য নির্বাচন করতে ব্যবহৃত একটি ভ্যারিয়েবল অথবা ভ্যালু।

এক্সসেপশন (exception): একটি রান-টাইম ত্রুটি।

থ্রো (throw): একটি এক্সসেপশন তৈরি করে।
স্ট্যাক ট্রেস (stack trace): যখন একটি এক্সসেপশন ঘটে তখন একটি প্রতিবেদন দেয়া যাতে প্রোগ্রামের বর্তমান অবস্থা প্রতীয়মান হয়।

প্রোটোটাইপ (prototype): একটি মেথডের প্রথম লাইন, যাতে সুনির্দিষ্ট ভাবে নাম, প্যারামিটার এবং রিটার্ন টাইপ বলা থাকে।

ট্র্যাভার্স (traverse): একটি সেটের সকল উপাদানে একই ধরনের অপারেশন ঘটিয়ে ভ্রমন করা।

কাউন্টার (counter): কোন কিছু গণনা করার জন্য ব্যবহৃত ভ্যারিয়েবল, সাধারণভাবে এর প্রাথমিক অবস্থা শূন্য এবং এটি আস্তে আস্তে বৃদ্ধি পেতে থাকে।

ইনক্রিমেন্ট (increment): ভ্যারিয়েবলের ভ্যালু এক করে বাড়ানো। জাভার ইনক্রিমেন্ট অপারেটর ++

ডিক্রিমেন্ট (decrement): ভ্যারিয়েবলের ভ্যালু এক করে কমানো। জাভার ডিক্রিমেন্ট অপারেটর –

৮.১২ অনুশীলনী (Exercises):

অনুশীলনী ৮.১: এমন একটি মেথড লিখ যা আর্গুমেন্ট হিসেবে একটি স্ট্রিং গ্রহন করে এবং স্ট্রিং গুলো উল্টোভাবে একই লাইনে প্রিন্ট করে।

অনুশীলনী ৮.২: ৮.৪ অংশের স্ট্যাক ট্রেস দেখ এবং নিচের প্রশ্নগুলোর উত্তর দাও:

- কি ধরনের এক্সসেপশন ঘটেছে, এবং কোন প্যাকেজে এটি সংজ্ঞায়িত?
- ইনডেক্স (যা এক্সসেপশন ঘটিয়েছে) এর ভ্যালু কত?
- কোন মেথড এক্সসেপশন ঘটিয়েছে, এবং মেথডটি কোথায় লিখা হয়েছে?
- কোন মেথড charAt কে আহ্বান করেছে?
- BadString.java তে, যেখানে charAt কে আহ্বান করা হয়েছে সেখানের লাইন নাম্বার কত?

অনুশীলনী ৮.৩: ৮.৭ অংশের কোড এমন একটি মেথডে আবদ্ধ কর যার নাম countLetters, এবং সর্বজনীন কর যাতে এটি স্ট্রিং এবং বর্ণ কে (letter) আর্গুমেন্ট হিসেবে গ্রহণ করতে পারে।

তারপর মেথডটি পুনরায় এমন ভাবে লিখ যাতে এটি একটার পর একটা ক্যারেক্টার পরীক্ষা না করে indexOf ব্যবহার করে a কে চিহ্নিত করতে পারে।

অনুশীলনী ৮.৪: এই অনুশীলনের উদ্দেশ্য হচ্ছে encapsulation এবং generalization রিভিউ করা।

1. নিচের কোড অংশটুকু encapsulate কর, এমন ভাবে এটিকে transform কর যাতে এটি আর্গুমেন্ট হিসেবে একটি স্ট্রিং গ্রহণ করে এবং count এর চূড়ান্ত মান রিটার্ন করে।

2. চূড়ান্ত ফলাফল প্রকাশকারী মেথড কি কাজ করে তা অল্প কথায় লিখ (বিস্তারিত লিখার প্রয়োজন নেই)।

3. এখন যেহেতু তুমি কোড generalize করে ফেলেছ সুতরাং এটি যে কোন স্ট্রিং এর জন্য কাজ করবে, এটিকে আরও সার্বজনীন (generalize) করার জন্য কি করা যেতে পারে?

```
String s = "((3 + 7) * 2)";
int len = s.length();

int i = 0;
int count = 0;

while (i < len) {
        char c = s.charAt(i);
        if (c == '(') {
                count = count + 1;
        } else if (c == ')') {
                count = count - 1;
        }
        i = i + 1;
}
        System.out.println(count);
```

অনুশীলনী ৮.৫: এই অনুশীলনের লক্ষ্য হচ্ছে জাভা টাইপ বিশ্লেষণ করা এবং কিছু জিনিস নিয়ে কাজ করা যা এই অধ্যায়ে আলোচনা করা হয়নি।

1. Test.java নামে একটি নতুন প্রোগ্রাম তৈরি কর এবং একটি main মেথড লিখ যার ভেতর বিভিন্ন ধরনের এক্সপ্রেশন + অপারেটর দ্বারা একত্রিত থাকবে। উদাহরণস্বরূপ, কী ঘটে যখন তুমি একটি স্ট্রিং এবং একটি ক্যারেক্টার "add" কর? তখন কি addition হয় নাকি concatenation হয়? ফলাফলের ধরন কি? (তুমি কীভাবে ফলাফলের ধরন বুঝতে সক্ষম হবে?)

2. নিচের টেবিলের একটি বড় আকারের অনুলিপি তৈরি কর এবং পূরণ কর। টেবিলে প্রতি জোড়ার মিলিত

অংশে, তোমরা লিখবে এই ধরনের সাথে + অপারেটর ব্যবহার বৈধ কি না, কোন ধরনের অপারেশন সংঘটিত হয়েছে (addition অথবা concatenation), এবং ফলাফলের ধরন কি।

	boolean	char	int	String
boolean				
char				
int				
String				

3. জাভার ডিজাইনাররা এই টেবিল পূরণ করার সময় কীভাবে বাছাই করেছিল তা চিন্তা কর। কতগুলো এন্ট্রি বাছাই করার অপশন না থাকার কারণে এড়ানো সম্ভব নয়? একাধিক যুক্তিসঙ্গত সম্ভাবনা থেকে অবাধে পছন্দ করার মত কতগুলো আছে? কতগুলো সন্দেহযুক্ত?

4. একটি ধাঁধা: সাধারণত, x++ স্টেটমেন্ট দ্বারা বোঝায় এটি x = x +1 এর সমতুল্য। কিন্তু যদি x একটি ক্যারেক্টার হয় (char) তাহলে এটি বোঝায় না! এই ক্ষেত্রে x++ বৈধ, কিন্তু x = x+1 ত্রুটির সৃষ্টি করে। চেষ্টা করে দেখ এবং খুঁজে বের কর ত্রুটির বার্তাটি কি। তারপর তুমি দেখ, কি ঘটছে তার সমাধান করতে পার কিনা।

অনুশীলনী ৮.৬: এই প্রোগ্রামের আউটপুট কি? mystery কি কাজ করে (কীভাবে) তা এক বাক্যে লিখ।

```
public class Mystery {

    public static String mystery(String s) {
        int i = s.length() - 1;
        String total = "";

        while (i >= 0 ) {
            char ch = s.charAt(i);
            System.out.println(i + "        " + ch);

            total = total + ch;
            i--;
        }
        return total;
    }
    public static void main(String[] args) {
        System.out.println(mystery("Allen"));
    }
}
```

অনুশীলনী ৮.৭: তোমার কোন বন্ধু তোমাকে নিচের মেথডটি দেখাল এবং ব্যাখ্যা করল যদি সংখ্যাটি হয় দুই অঙ্কের, তাহলে প্রোগ্রামটি আউটপুট দিবে উল্টো করে। সে দাবী করল, যদি সংখ্যাটি হয় 17 তাহলে আউটপুট

দেখাবে 71।

সে কি সঠিক? যদি তা না হয়, ব্যাখ্যা কর প্রোগ্রামটি আসলে কী করে এবং সঠিক জিনিস করার জন্য এটিকে পরিবর্তন কর।

```
int number = 17;
int lastDigit = number%10;
int firstDigit = number/10;
System.out.println(lastDigit + firstDigit);
```

অনুশীলনী ৮.৮: নিচের প্রোগ্রামটির আউটপুট কি?

```
public class Enigma {

    public static void enigma(int x) {
        if (x == 0) {
            return;
        } else {
            enigma(x/2);
        }
        System.out.print(x%2);
    }

    public static void main(String[] args) {
        enigma(5);
        System.out.println("");
        }
    }
```

৪ থেকে ৫ শব্দে ব্যাখ্যা কর enigma মেথডটি কি আসলে কাজ করে।

অনুশীলনী ৮.৯:

১। একটি নতুন জাভা প্রোগ্রাম তৈরি কর এবং নাম দাও Palindrome.java।

২। first নামক একটি মেথড লিখ যা একটি স্ট্রিং গ্রহণ করবে এবং প্রথম বর্ণ রিটার্ন করবে, এবং আরেকটি মেথড লিখ last নামে যা সর্বশেষ বর্ণ রিটার্ন করবে।

৩। middle নামক একটি মেথড লিখ যা একটি স্ট্রিং গ্রহণ করবে এবং একটি সাবস্ট্রিং রিটার্ন করবে যাতে প্রথম এবং শেষ ক্যারেক্টার ছাড়া সব বিদ্যমান থাকবে।

ইঙ্গিত: স্ট্রিং ক্লাসে সাবস্ট্রিং মেথডের ডকুমেন্টেশন পড়। নিশ্চিতভাবে উপলব্ধি করার জন্য middle লিখতে চেষ্টা করার পূর্বে কিছু পরীক্ষামূলক প্রোগ্রাম চালনা কর।

কি ঘটে যখন তুমি স্ট্রিং এ একটি middle কে আহ্বান কর যার মাত্র দুটি বর্ণ রয়েছে? একটি বর্ণ রয়েছে? কোন বর্ণ

যখন থাকে না?

৪। palindrome এর সাধারণ সংজ্ঞা হচ্ছে একটি শব্দ যা শুরু থেকে এবং শেষ থেকে পড়লে একই হয়। যেমন: "otto" এবং "palindromeemordnilap" ইত্যাদি। একটি বিকল্প এ ধরনের বৈশিষ্ট্য ডিফাইন করার বিকল্প উপায় হচ্ছে এই বৈশিষ্ট্য কিভাবে পরীক্ষা করা হবে তা সুনির্দিষ্ট করা। উদাহরণস্বরূপ, আমরা বলতে পারি "এক অক্ষরের শব্দ একটি palindrome, এবং দুই অক্ষরের শব্দ একটি palindrome যদি তারা একই অক্ষর হয়, এবং যে কোন শব্দ palindrome হবে যদি তার প্রথম এবং শেষ অক্ষর একই হয় এবং মাঝখানেরগুলো palindrome হয়"।

isPalindrome নামক একটি রিকারসিভ মেথড লিখ যা একটি স্ট্রিং গ্রহণ করবে এবং একটি বুলিয়ান রিটার্ন করবে। যার মাধ্যমে জানা যাবে শব্দটি palindrome কিনা।

৫। যখন তোমার একটি সক্রিয় palindrome চেকার তৈরি হয়ে যাবে তার পর চেষ্টা কর যে এই প্রোগ্রামে ব্যবহৃত একাধিক শর্তযুক্ত মেথড কমিয়ে আনা যায় কিনা।

ইঙ্গিত: এটা জেনে রাখা ভাল যে একটি empty (খালি) স্ট্রিং হচ্ছে একটি palindrome।

৬। একটুকরো কাগজে, palindrome পুনরাবৃত্তি করার কৌশল খুঁজে বের কর। এটার অনেকগুলো উপায় রয়েছে। সুতরাং কোড লিখার পূর্বে নিশ্চিত হয়ে নাও তোমার পরিকল্পনাটি দৃঢ়।

৭। isPalindromeIter নামক মেথডে তোমার কৌশলটি বাস্তবায়ন কর।

৮। ঐচ্ছিক: অ্যাপেনডিক্স B একটি ফাইল হতে শব্দের তালিকা পড়ার জন্য কোড প্রদান করে। শব্দের তালিকাটি পড় এবং palindrome গুলো প্রিন্ট কর।

অনুশীলনী ৮.১০: একটি শব্দকে "abecedarian" (বর্ণানুক্রমে সজ্জিত) বলা হবে যদি শব্দটির বর্ণগুলো বর্ণমালার ক্রম অনুযায়ী দৃশ্যমান হয়। উদাহরণস্বরূপ, নিচের শব্দগুলো সবই ৬ বর্ণ বিশিষ্ট abecedarian শব্দ।

> abdest, acknow, acorsy, adempt, adipsy, agnosy, befist, behint,
> beknow, bijoux, biopsy, cestuy, chintz, deflux, dehors, dehort,
> deinos, diluvy, dimpsy

১। একটি প্রক্রিয়ার বর্ণনা দাও যার মাধ্যমে তুমি পরীক্ষা করতে পারবে প্রদত্ত শব্দ abecedarian, এটা মনে রাখবে শব্দগুলোর সব বর্ণই ছোটহাতের। তোমার প্রক্রিয়াটি হতে পারে iterative অথবা recursive।

২। isAbecedarian নামক মেথডে তোমার প্রক্রিয়াটি বাস্তবায়ন কর।

অনুশীলনী ৮.১১: dupledrome হচ্ছে এমন শব্দ যা শুধুমাত্র দ্বৈত শব্দ ধারণ করে, যেমন "llaammaa" অথবা "ssaabb"। আমি ধারণা করি সাধারণ ইংরেজী ব্যবহারে কোন dupledrome নেই। আমার অনুমানটি পরীক্ষা করতে আমি একটি প্রোগ্রাম পছন্দ করব যা অভিধান থেকে এক এক করে শব্দ নিয়ে পড়বে এবং পরীক্ষা করবে তার dupledromity.

IsDupledrome নামক একটি মেথড লিখ যা একটি স্ট্রিং গ্রহণ করবে এবং একটি বুলিয়ান রিটার্ন করবে এবং বলে

দিবে তা dupledrome কিনা।

অনুশীলনী ৮.১২:
১। Captain Crunch Decoder ring কাজ করে একটি একটি করে বর্ণ গ্রহণ করার মাধ্যমে এবং এর সাথে ১৩ যোগ করে। উদাহরণস্বরূপ, “a” হয়ে যাবে “n” এবং “b” হয়ে যাবে “o”। বর্ণগুলো শেষে গিয়ে আবার ঘুরে আসে, সুতরাং “z” হয়ে যায় “m”

একটি মেথড লিখ যা একটি স্ট্রিং গ্রহণ করবে এবং একট নতুন স্ট্রিং রিটার্ন করবে যাতে এনকোডেড সংস্করণ বিদ্যমান থাকবে। তোমার ধরে নেয়া উচিৎ যে, স্ট্রিং বড় হাতের এবং ছোট হাতের অক্ষর এবং স্পেস ধারণ করবে, কিন্তু কোন যতিচিহ্ন ধারণ করবে না। ছোট হাতের অক্ষর পরিবর্তিত হয়ে অন্য একটি ছোট হাতের অক্ষরে পরিনত হবে, বড় হাতের অক্ষর বড় হাতের অক্ষরে পরিনত হবে। তুমি অবশ্যই স্পেস এনকোড করবে না।

২। Captain Crunch মেথডকে সার্বজনীন কর যাতে এটি ১৩ যোগ করার পরিবর্তে যে কোন প্রদত্ত সংখ্যা গ্রহণ করে যোগ করে। এখন তুমি ১৩ যোগ করে এনকোড করতে পারবে এবং -১৩ যোগ করে ডিকোড করতে পারবে। চেষ্টা কর।

অনুশীলনী ৮.১৩: তুমি যদি অধ্যায় ৫ এ গ্রিডওয়ার্ল্ড অনুশীলনীটি করে থাক, তুমি হয়ত এই অনুশীলনীটি উপভোগ করবে। লক্ষ্য হচ্ছে ত্রিকোণমিতি ব্যবহার করে Bug গুলোর একটা আরেকটার পিছু লাগানো (chasing)।

BugRunner.java অনুলিপি কর এবং নাম দাও ChaseRunner.java এবং এটিকে তোমার ডেভেলপমেন্ট পরিবেশে ইমপোর্ট কর। কোন কিছু পরিবর্তন করার পূর্বে, পরীক্ষা কর তুমি এটি কম্পাইল এবং চালাতে পারবে কিনা।

- দুটি Bug তৈরি কর, একটি লাল এবং অন্যটি নীল।

- distance নামক একটি মেথড লিখ যা দুটি বাগ গ্রহণ করে এবং তাদের মধ্যবর্তী দূরত্ব গণনা করে। মনে রাখবে যাতে তুমি বাগ এর x-coordinate পাও। এরকম:

```
int x = bug.getLocation().getCol();
```

- turnToward নামক মেথড লিখ যা দুটি বাগ গ্রহণ করে এবং একটিকে অন্যটির মুখোমুখি করে। ইঙ্গিত: Math.atan2 ব্যবহার কর, তবে মনে রাখবে এর ফলাফল হবে রেডিয়ানে (radians), সুতরাং তোমার এটিকে ডিগ্রি তে রূপান্তর করে নিতে হবে। এছাড়াও, বাগ এর জন্য ০ ডিগ্রি হচ্ছে উত্তর দিক, পূর্ব দিক নয়।

- moveToward নামক একটি মেথড লিখ যা দুটি বাগ গ্রহণ করবে, প্রথমটিকে দ্বিতীয়টির মুখোমুখি করবে, এবং তারপর প্রথমটি সরে যাবে, যদি সম্ভব হয়।

- moveBugs নামক একটি মেথড লিখ যা দুটি বাগ গ্রহণ করবে এবং সেই সাথে গ্রহণ করবে একটি পূর্ণসংখ্যা n, এবং প্রতিটি বাগ অন্য n সময়ের দিকে যেতে থাকবে। তুমি এই মেথড রিকারসিভ ব্যবহার করে লিখতে পার, অথবা একটি while লুপ ব্যবহার করতে পার।

- প্রতিটি মেথড ডেভেলপ করার পর পরীক্ষা কর। যখন দেখবে তারা সব কাজ করছে তখন এই মেথড গুলোর উন্নয়ন করা সম্ভব কিনা তা খুঁজে দেখ। উদাহরণস্বরূপ, যদি distance এবং turnToward মেথডে প্রয়োজনাতিরিক্ত কোড থাকে তাহলে তুমি একাধিকবার ব্যবহৃত কোড একটি মেথডে আবদ্ধ কর।

অধ্যায় ৯

পরিবর্তনযোগ্য অবজেক্ট (Mutable objects)

Strings হলো অবজেক্ট কিন্তু তারা সাধারণ বা স্বাভাবিক অবজেক্ট নয় কারণ-

- তারা অপরিবর্তনীয় (immutable)
- তাদের কোন গুণাবলী (attributes) নেই
- এর একটি তৈরি করার জন্য আমাদের new ব্যবহার করতে হয় না

এই অধ্যায়ে আমরা জাভা লাইব্রেরি থেকে দুটি অবজেক্ট ব্যবহার করব। আর এগুলো হলো Point এবং Rectangle। কিন্তু প্রথমেই আমাদের একটি বিষয় পরিষ্কার হয়ে নিতে হবে আর তা হলো এই Point এবং Rectangle স্ক্রীনে দৃশ্যমান কোন চিত্রভিত্তিক বা graphical অবজেক্ট নয়। এগুলো হলো অনেকটা ints এবং doubles এর মতো values যা ডেটা ধারণ করে। অন্য values এর মতো তারা অন্তঃস্থভাবে ব্যবহৃত হয় গণনা করার জন্য।

৯.১ প্যাকেজসমূহ (Packages)

জাভা লাইব্রেরিগুলো বিভিন্ন প্যাকেজে বিভক্ত। যার মধ্যে রয়েছে java.lang এবং java.awt। java.lang আমরা যত ধরনের classes ব্যবহার করি তার সবগুলোই ধারণ করে। আর java.awt হলো Abstract Window Toolkit (AWT), যা windows, বাটন (button), গ্রাফিক্সের (graphics) জন্য classes ধারণ করে।

অন্য প্যাকেজে বর্ণিত কোন class ব্যবহার করার জন্য আমাদের তা import করতে হবে। Point এবং Rectangle এই দুটি রয়েছে java.awt প্যাকেজে, তাই একে import করার জন্য আমাদের নিচের কোড লিখতে হবে:

```
import java.awt.Point;
import java.awt.Rectangle;
```

সব ধরনের import statements প্রোগ্রাম শুরুর আগে এবং class বর্ণনার আগেই লিখতে হয়।

java.lang এর অন্তর্ভুক্ত classes যেমন- Math এবং String স্বয়ংক্রিয়ভাবে import হয়ে যায়, তাই এখন পর্যন্ত আমাদের import statement লেখার প্রয়োজন হয়নি।

৯.২ পয়েন্ট অবজেক্ট (Point Objects)

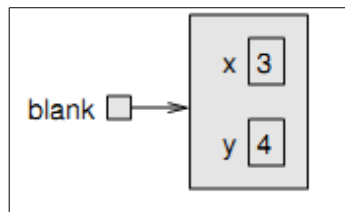
একটি পয়েন্ট (point) হলো দুটি নম্বর বা স্থানাংক (coordinates), যাকে আমরা সম্মিলিতভাবে একটি একক অবজেক্ট হিসেবে চিন্তা করতে পারি। গাণিতিক নিয়ম অনুসারে, points সবসময় বন্ধনীর (parentheses) ভিতরে লেখা হয় এবং দুটি স্থানাঙ্ককে (coordinates) কমা দিয়ে আলদা করা হয়। উদাহরণস্বরূপ বলা যায়, (0; 0) এটি

নির্দেশ করে মূল বিন্দু এবং (x; y) নির্দেশ করে এমন একটি বিন্দু যা point x ইউনিট ডানদিক এবং y ইউনিট মূলবিন্দুর ওপরের অবস্থান বোঝায়।

জাভাতে একটি পয়েন্ট point object দ্বারা উপস্থাপিত হয়। একটি new তৈরির জন্য আমাদের নতুন একটি ব্যবহার করতে হয়:

```
Point blank;
blank = new Point(3, 4);
```

প্রথম লাইন হলো গতানুগতিকভাবে ভ্যারিয়েবল declaration: blank আমাদের জন্য পয়েন্ট টাইপ করবে। দ্বিতীয় লাইন নতুন অবজেক্টকে ডাকবে (invoke) এবং এর ধরন নির্দেশ করবে এবং আর্গুমেন্ট (arguments) প্রদান করবে। এই আর্গুমেন্টই হলো নতুন পয়েন্টের কো-অর্ডিনেটর (coordinates), (3; 4)।

নতুন পয়েন্টের reference হলো new এর ফলাফল। তাই blank ধারণ করবে নতুন তৈরিকৃত অবজেক্টের reference। এখানে এই অ্যাসাইনমেন্টের আঁকার আদর্শ উপায় নিচের চিত্রের সাহায্যে দেখানো হলো।

সাধারণভাবে, ভ্যারিয়েবলের নাম "blank" বাক্সের বাইরে থাকবে এবং এর value থাকবে বক্সের ভিতরে। এক্ষেত্রে value হলো reference, যা চিত্রে তীর চিহ্নের সাহায্যে দেখানো হয়েছে। এই তীর চিহ্ন অবজেক্টকে পয়েন্টের দিকে referrer করে।

দুটি values সহকারে নতুন তৈরিকৃত অবজেক্ট আমরা বক্স-এ দেখতে পাচ্ছি। x এবং y-এই নাম দুটি হলো instance variables এর নাম। সব যদি আমরা একসাথে চিন্তা করি তাহলে বলা যায়, একটি প্রোগ্রামের সব variables, values, এবং objects কে একসাথে state বলে। উপরের ড্রায়াগ্রামের মতো যেগুলো আমাদের প্রোগ্রামের অবস্থা বা state বুঝতে সাহায্য করে তাকে state diagrams বলে। যখন প্রোগ্রাম চলতে থাকবে তখন state এর পরিবর্তন হবে। তাই আমাদের state diagram এ একটি নির্দিষ্ট পয়েন্টে চালুর বা শুরুর অবস্থান ঠিক করে করতে হবে snapshot এর মতো।

৯.৩ ইন্সট্যান্স ভ্যারিয়েবল (Instance variables)

এক গুচ্ছ ডেটা বা (pieces of data) যা একটি অবজেক্ট তৈরি করে তাকে বলে instance variables, কারণ প্রত্যেকটি অবজেক্টই তার ধরন অনুসারেই instance এবং এদের প্রত্যেকেরই নিজেদের instance variables এর কপি রয়েছে।

এটি অনেকটা গাড়ির ড্যাশবক্স (glove compartment) মতো। প্রত্যেকটি গাড়ি বা car ই " car" এর একটি instance type এবং প্রত্যেকটি গাড়িরই নিজস্ব ড্যাশবক্স রয়েছে। যদি আমাকে কেউ বলে তার গাড়ির ড্যাশবক্স থেকে কোন কিছু নিতে তাহলে প্রথমে আমাকে বলতে হবে তার গাড়ি কোনটি।

একই ভাবে যদি আমরা কোন instance variable থেকে value পড়তে চাই তাহলে আমাদের অবশ্যই সুনির্দিষ্ট করতে হবে কোন অবজেক্ট আমরা নিতে চাই। জাভাতে এই কাজটি করা হয় "dot notation" ব্যবহার করে।

```
int x = blank.x;
```

এই এক্সপ্রেশনের "blank.x" বলতে বোঝায় "refers করার জন্য blank অবজেক্টে যাও, এবং x-এর value নাও।" এক্ষেত্রে আমরা x নামক local variable এ value অ্যাসাইন (assign) করতে পারি। x নামক local variable এবং x নামক instance variable এর মধ্যে কোন দ্বন্দ্ব নেই। "dot notation" এর উদ্দেশ্য হলো অস্পষ্টভাবে যেসব ভ্যারিয়েবল referrer করা হয়ছে তা নির্দিষ্ট করা।

যে কোন জাভা expression এ আমরা "dot notation" ব্যবহার করতে পারি। তাই নিচের কোডগুলো যৌক্তিক।

```
System.out.println(blank.x + ", " + blank.y);
int distance = blank.x * blank.x + blank.y * blank.y;
```

প্রথম লাইন প্রিন্ট করবে 3, 4; দ্বিতীয় লাইন গণনা করে value 25 বের করবে।

৯.৪ প্যারামিটার হিসেবে অবজেক্ট (Objects as parameters)

সাধারণ নিয়মানুসারেই আমরা প্যারামিটার হিসেবে objects pass করতে পারি। উদাহরণস্বরূপ,

```
public static void printPoint(Point p) {
        System.out.println("(" + p.x + ", " + p.y + ")");
}
```

এই method, argument হিসেবে একটি পয়েন্ট নিবে এবং প্রিন্ট একে একটি আর্দশ ফরমেটে প্রিন্ট করবে। যদি আমরা printPoint(blank) কে invoke করি তাহলে তা (3, 4) প্রিন্ট করবে। আসলে, জাভাতে পয়েন্ট প্রিন্ট করার জন্য method সুনির্দিষ্ট করাই আছে। যদি আমরা invoke করি; System.out.println(blank), তাহলে আমরা পাবো,

```
java.awt.Point[x=3,y=4]
```

জাভাতে এটি হলো অবজেক্ট প্রিন্ট করার আর্দশ ফরমেট। এটি ধরনের নাম প্রিন্ট করে যা instance variables এর নাম এবং values অনুসরণ করে।

দ্বিতীয় উদাহরণ হিসেবে, সেকশন ৬.২ এর distance method কে আমরা পুনরায় লিখতে পারি। যার ফলে, এটি প্যারামিটার হিসেবে চারটি doubles এর পরিবর্তে এটি দুটি পয়েন্ট নিবে।

```
public static double distance(Point p1, Point p2) {
        double dx = (double) (p2.x - p1.x);
        double dy = (double) (p2.y - p1.y);
        return Math.sqrt(dx*dx + dy*dy);
}
```

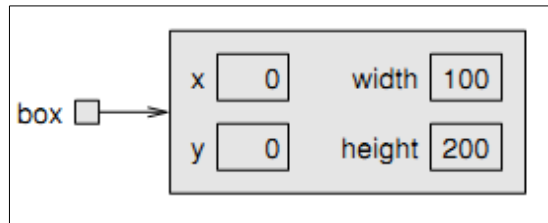
এখানে typecasts এর আসলে প্রয়োজনীয় নয়; আমি তাদের reminder হিসেবে সংযুক্ত করতে পারি যেখানে instance variables এর মধ্যে রয়েছে পয়েন্ট যা integers।

৯.৫ আয়তক্ষেত্রে (Rectangles)

আয়তক্ষেত্রগুলো অনেকটা পয়েন্টের মতোই। পার্থক্য শুধুমাত্র আয়তক্ষেত্রের রয়েছে চারটি instance variables: x, y, width (প্রস্থ) এবং height (উচ্চতা)। এই পার্থক্য ছাড়া বাকিগুলো বেশিরভাগই এক রকম।

এই উদাহরণটি একটি আয়তকার অবজেক্ট তৈরি করবে এবং একটি বাক্সকে refer করবে।

```
Rectangle box = new Rectangle(0, 0, 100, 200);
```

নিচের চিত্রটি আমাদের দেখাবে এই অ্যাসাইনমেন্টের প্রভাব।

যদি আমরা বাক্সটিকে প্রিন্ট করি তাহলে আমরা পাবো,

```
java.awt.Rectangle[x=0, y=0, width=100, height=200]
```

পুনরায় বলতে হচ্ছে, এটি জাভার একটি method যা জানে কীভাবে একটি আয়তক্ষেত্র প্রিন্ট করতে হয়।

৯.৬ রিটার্ন টাইপ হিসেবে অবজেক্ট (Objects as return types)

আমরা এমন methods ও লিখতে পারি যা অবজেক্ট return করবে। উদাহরণস্বরূপ, findCenter আয়তক্ষেত্রেকে (Rectangle) আর্গুমেন্ট হিসেবে নেয় এবং এমন পয়েন্ট রিটার্ন করে যা ধারণ করে আয়তক্ষেত্রের কেন্দ্রস্থিত কোর্ডিনেটর।

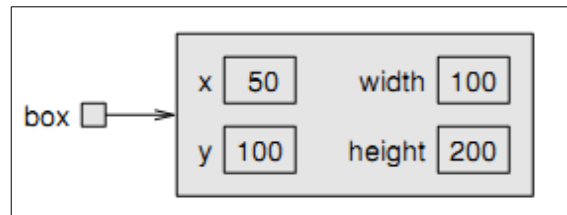
```
public static Point findCenter(Rectangle box) {
        int x = box.x + box.width/2;
        int y = box.y + box.height/2;
        return new Point(x, y);
}
```

আমাদের মনে রাখতে হবে, আমরা new ব্যবহার করতে পারি নতুন অবজেক্ট তৈরি করার জন্য এবং তারপর সাথে সাথেই রিটার্ন value হিসেবে ফলাফল ব্যবহার করতে পারি।

৯.৭ অবজেক্ট পরিবর্তনযোগ্য (Objects are mutable)

আমরা যেকোন অবজেক্ট এর বিষয়বস্তু পরিবর্তন করতে পারি। এর জন্য আমাদের সেই অবজেক্টের কোন একটি instance variables এর assignment তৈরি করতে হবে। উদাহরণস্বরূপ, 'একটি আয়তক্ষেত্রকে তার আকার পরির্বতন না করে "move" করার জন্য আমরা x and y values পরির্বতন করতে পারি:

```
box.x = box.x + 50;
box.y = box.y + 100;
```

এর ফলাফল আমরা নিচের চিত্রে দেখতে পারবো:

আমরা এই কোডকে একটি method এর মধ্যে এনক্যাপসুলেট (encapsulate) করতে পারি এবং যে কোন ভাবেই সর্বজনীনকরণের (generalize) মাধ্যমে আয়তক্ষেত্রকে move করাতে পারি:

```
public static void moveRect(Rectangle box, int dx, int dy) {
        box.x = box.x + dx;
        box.y = box.y + dy;
}
```

dx এবং dy নির্দেশ করে কোন দিকে ঠিক কতটা দূরুত্ব পর্যন্ত আমাদের আয়তক্ষেত্রকে move করাতে হবে। এই method কে invoke করার সময়, যা আমরা আর্গুমেন্ট হিসেবে pass করবো তা আমাদের আয়তক্ষেত্রের পরিবর্তনের ওপর প্রভাব ফেলবে।

```
Rectangle box = new Rectangle(0, 0, 100, 200);
moveRect(box, 50, 100);
System.out.println(box);
```

prints java.awt.Rectangle[x=50,y=100,width=100,height=200]

Method এর মধ্যে আর্গুমেন্ট হিসেবে অবজেক্ট pass করে তাদের পরিবর্তন করা ব্যবহারযোগ্য হতে পারে কিন্তু এটি সাথে সাথে ডিবাগিং-কে (debugging) আরও বেশি জটিল করে তোলে। কারণ এটা কোন সময়ই বোঝা যায় না কোন method invocation করে অথবা কোন method তাদের আর্গুমেন্ট পরিবর্তন করে। পরে আমরা এধরনের প্রোগ্রামিং ল্যাঙ্গুয়েজের সুবিধা আর অসুবিধা নিয়ে আলোচনা করবো।

জাভা আমাদের এমন methods নিয়ে কাজ করতে দেয় যা Points এবং Rectangles ওপর অপারেট করে। আমরা এই নথিগুলো পড়তে পারবো নিচের লিংকগুলো থেকে;
http://download.oracle.com/javase/6/docs/api/java/awt/Point.html এবং

http://download.oracle.com/javase/6/docs/api/java/awt/Rectangle.html।

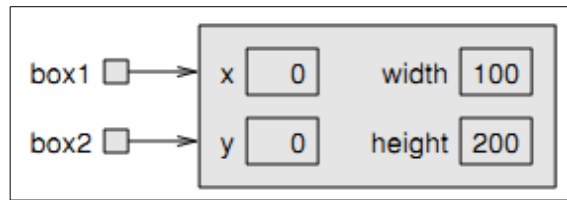
উদাহরণস্বরূপ, translate এর প্রভাব moveRect এর মতো। কিন্তু আয়তক্ষেত্রকে আর্গুমেন্ট হিসেবে pass করার চেয়ে আমরা dot notation ব্যবহার করতে পারি:

```
box.translate(50, 100);
```

৯.৮ অ্যালিয়াসিং (Aliasing)

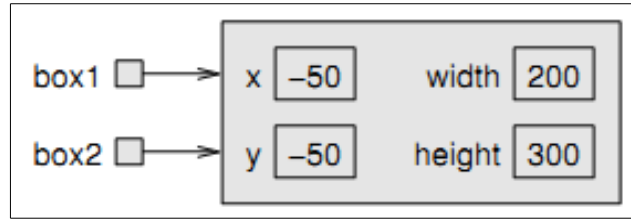
মনে রাখতে হবে যখন আমরা কোন ভ্যারিয়েবলে অবজেক্ট অ্যাসাইন (assign) করি, তখন আমরা মূলত অবজেক্টে reference অ্যাসাইন করি। অনেকগুলো ভ্যারিয়েবল দিয়ে একটি অবজেক্টকে refer করাও সম্ভব। উদাহরণস্বরূপ, এই কোড:

```
Rectangle box1 = new Rectangle(0, 0, 100, 200);
Rectangle box2 = box1;
```

একটি state diagram উৎপন্ন করবে যার স্বরূপ হবে:

box1 এবং box2 একই অবজেক্টকে refer করছে। অন্য অর্থে বলা যায়, এই অবজেক্টের দুটি নাম, box1 এবং box2। যখন কোন ব্যক্তি দুটি নাম ব্যবহার করেন তখন তাকে বলে aliasing। একই ব্যাপার অবজেক্টের জন্য।

যখন দুটি ভ্যারিয়েবল aliased হয়, একটি ভ্যারিয়েবলের যেকোন পরিবর্তনের প্রভাব অন্য ভ্যারিয়েবলের ওপরও পরে। উদাহরণস্বরূপ:

```
System.out.println(box2.width);
box1.grow(50, 50);
System.out.println(box2.width);
```

এক্ষেত্রে প্রথম লাইন প্রিন্ট করবে 100, যা আয়তক্ষেত্রের প্রস্থ এবং এটি box2 দ্বারা referred করা হয়েছে। দ্বিতীয় লাইনটি box1 এর grow method কে invoke করবে, যা আয়তক্ষেত্রকে প্রত্যেক দিকে 50 pixels বাড়াবে। (আরও বিস্তারিত জানার জন্য নথিসমূহ দেখতে হবে)। এই প্রভাব চিত্রে দেখা যাচ্ছে:

যাই হোক না কেন, box1 যে ধরনের পরিবর্তন হবে তার সবটাই box2 এর জন্য প্রযোজ্য হবে। তাই, তৃতীয় লাইন যে value প্রিন্ট করবে তা হবে 200, অর্থাৎ বিস্তৃত আয়তক্ষেত্রের প্রস্থ হবে 200। (এর পাশাপাশি, আয়তক্ষেত্রের কোর্ডিনেটর ঋণাত্মক হওয়া খুব স্বাভাবিক)

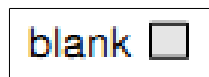
আগের মতোই আমরা এই সাধারণ উদাহরণের জন্যও বলতে পারি, যেসব কোড aliasing এর সাথে যুক্ত তাদের নিয়ে দ্বিধা তৈরি হতে পারে এবং এগুলোর ডিবাগিং করা জটিল হতে পারে। সাধারণভাবে বলা যায়, aliasing ব্যবহার করা হয় আমাদের পরিহার করতে হবে অথবা অনেক সাবধানে ব্যবহার করতে হবে।

৯.৯ ফাঁকা (Null)

যখন আমরা একটি অবজেক্ট ভ্যারিয়েবল তৈরি করবো, তখন মনে রাখতে হবে আমরা একটি অবজেক্টের reference তৈরি করছি। যতক্ষণ না পর্যন্ত আমরা ভ্যারিয়েবলটিকে একটি অবজেক্ট এর দিকে point না করাচ্ছি। এই ভ্যারিয়েবলের value হলো null। null হলো একটি বিশেষ value (এবং একটি জাভা কিওয়ার্ড) এর অর্থ "no object (অবজেক্ট শূন্য)"।

Point blank এর declaration (ডিক্লেয়ারেশন); এর সমানই হলো এই initialization

```
Point blank = null;
```

এবং এটি আমাদের এই state diagram দেখাবে:

Null এর value উপস্থাপিত হয় কোন তীর চিহ্ন (arrow) ছাড়া ছোট একটি বর্গক্ষেত্র (square) দিয়ে। যদি আমরা একটি null object ব্যবহার করতে চাই, তাহলে হয় আমাদের একটি instance variable একসেস (access) করতে হবে অথবা একটি method কে invoke করতে হবে, জাভার NullPointerException প্রদান করলে, তা একটি error message প্রিন্ট করবে এবং প্রোগ্রামটি বন্ধ (terminate) করে দেবে।

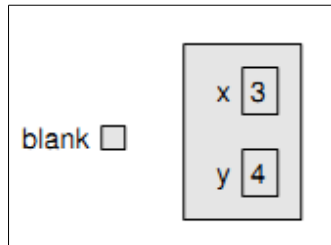
```
Point blank = null;
int x = blank.x;                    // NullPointerException
blank.translate(50, 50);            // NullPointerException
```

অন্যদিকে, আর্গুমেন্ট হিসেবে null object pass করা অথবা রিটার্ন value হিসেবে one গ্রহণ করা যুক্তিযুক্ত। এমনকি, এরকম করাও খুব সাধারণ ব্যাপার। উদাহরণস্বরূপ বলা যায়, খালি সেট বা উপস্থাপন করার জন্য বা error condition নির্দেশ করার জন্য আমরা এই কাজ করতে পারি।

৯.১০ গার্বেজ সংগ্রহসমূহ (Garbage collection)

সেকশন ৯.৮ এ, একই অবজেক্টকে একাধিক ভ্যারিয়েবল দিয়ে refer করলে কি হয় তা নিয়ে আমরা আলোচনা করছিলাম। আমরা কি ধারণা করতে পারি, যখন কোন অবজেক্টকে কোন ভ্যারিয়েবল refers করবে না তখন কি হবে? উদাহরণস্বরূপ:

```
Point blank = new Point(3, 4);
blank = null;
```

প্রথম লাইন তৈরি করবে একটি "new Point" অবজেক্ট (object) এবং একটি blank তৈরি করবে একে refer করার জন্য। দ্বিতীয় লাইন blank পরিবর্তন করবে যাতে করে কোন অবজেক্ট refer করার পরিবর্তে তা কোন কিছু কেই refers না করে। অর্থাৎ তা null object কে refer করে।

যদি কোন কিছু কোন অবজেক্টকে refer না করে, তারপর যদি কেউ এর values পড়তে বা লিখতে চায় অথবা এর ওপর কোন method invoke করতে চায়। এর ফলস্বরূপ, এই প্রোগ্রামটি exist হয়ে যাবে। আমরা সহজেই মেমরিতে অবজেক্ট রাখতে পারি, কিন্তু তাতে শুধুমাত্র আমাদের জায়গাই অপচয় করবে। তাই পর্যায়ক্রমে যখন আমাদের প্রোগ্রাম run করবে, তখন সিস্টেম stranded objects খুঁজবে এবং তাদের reclaim করবে। এই প্রক্রিয়াকে বলে garbage collection। পরবর্তিতে, মেমরির যে স্থান অবজেক্ট দ্বারা পূর্ণ বা ব্যবহৃত ছিলো তা new object এর অংশ হিসেবে ব্যবহার উপযোগী হবে।

Garbage collection তৈরি করার জন্য আমাদের কিছুই করতে হবে না এবং সাধারণভাবে আমাদের তার জন্য সচেতনও হতে হবে না। কিন্তু আমাদের জানতে হবে এটি পিছনে এটি পর্যায়ক্রমে চলতে থাকবে।

৯.১১ অবজেক্ট এবং প্রাইমেটিভ (Objects and primitives)

জাভাতে দুই প্রকার types রয়েছে, primitive types এবং object types। Primitives, হলো int এবং boolean যার শুরু হয় ছোট হাতের অক্ষর দ্বারা (lower-case letters); আর object types এর শুরু হয় বড় হাতের অক্ষর দ্বারা (upper-case letters)। নিচের পার্থক্যগুলো আমাদের জন্য প্রয়োজনীয় আর এগুলো আমাদের এদের পার্থক্য মনে রাখতে সাহায্য করবে:

- যখন আমরা কোন primitive variable declare করবো, তখন আমরা এই primitive value এর জন্য storage space বা জায়গা পাবো। আবার যখন আমরা object variable declare করবো তখন আমরা একটি space পাবো যা ব্যবহৃত হবে object এর reference নির্দেশ করার জন্য। অবজেক্ট এর নিজের জন্য space এর দরকার হলে আমাদের new ব্যবহার করতে হবে।

- যদি আমরা কোন primitive type কে initialize না করি, তাহলে একটি নির্ধারিত value নিয়ে নিবে এর type অনুসারে বা type এর ওপর নির্ভর করে। উদাহরণস্বরূপ, ints এর জন্য 0 এবং booleans এর

জন্য false। আর object types এর জন্য পূর্ব নির্ধারিত value হলো null, যা নির্দেশ করে no object।

- Primitive variables খুব স্পষ্টভাবেই পৃথক বা বিচ্ছিন্ন (isolated); এদের পৃথক বা বিচ্ছিন্ন বলার কারণ হচ্ছে আমরা একটি method এ এমন কোন পরিবর্তন করতে পারি না যা অন্য কোন method এর variable এর ওপর প্রভাব বিস্তার করবে।

- Object variables এর সাথে কাজ করার জন্য মাঝে মাঝে একটু কৌশলী হতে হয় কারণ তারা খুব ভালোভাবে বিচ্ছিন্ন নয়। যদি আমরা কোন object এ argument হিসেবে reference pass করি, তাহলে যে method আমরা invoke করি তা object কে modify করতে পারে। এক্ষেত্রে আমরা প্রভাবটি দেখতে পাবো। অবশ্যই এটি একটি ভালো দিক, কিন্তু আমাদের এ ব্যাপারে সচেতন হতে হবে।

এগুলো ছাড়া আর কোন পার্থক্য নেই primitives এবং object types এর মধ্যে। আমরা জাভাতে কোন নতুন primitives সংযুক্ত করতে পারি না (যদি না আমরা আমাদের standards committee এর সদস্য করতে পারি), কিন্তু আমরা নতুন object types তৈরি করতে পারি। সেটি কীভাবে করতে হবে তা আমরা পরবর্তী অধ্যায়ে দেখতে পারবো।

৯.১২ শব্দকোষ (Glossary)

package: কতগুলো classes এর সমন্বয়। জাভার classes গুলো packages এর মধ্যে বিন্যস্ত থাকে।

AWT: AWT হলো Abstract Window Toolkit, এটি হলো জাভার packages সমূহের মধ্যে বড় এবং অন্যতম ব্যবহৃত Java packages.

instance: এটি হলো category এর একটি উদাহরণ। উদাহরণ হিসেবে বলা যায়, My cat হলো "feline things" এর category এর একটি instance। প্রত্যেক object হলো কিছু class এর instance।

instance variable: অবজেক্ট তৈরিতে ব্যবহৃত একটি নামসহ ডেটা items। প্রত্যেক অবজেক্ট (instance) এর নিজ নিজ class এর জন্য নিজস্ব instance variables এর অনুলিপি রয়েছে।

reference: অবজেক্ট নির্দেশক একটি value। state diagram এ, একটি reference arrow বা তীর চিহ্ন দিয়ে দেখানো হয়।

aliasing: দুই বা ততোধিক ভ্যারিয়েবল যখন এক অবজেক্টকে refer করে।

garbage collection: References ছাড়া যেসব অবজেক্ট আছে তাদের খুঁজে বের করা এবং অবজেক্টের storage space reclaiming করা।

state: প্রোগ্রাম execution এর সময় প্রদত্ত যে কোন পয়েন্টে সব ভ্যারিয়েবল এবং অবজেক্টের এর সব values সহ সম্পূর্ণ বর্ণনা।

state diagram: যে কোন প্রোগ্রামের একটি snapshot যা প্রোগ্রামের state দেখায়। আর এটি প্রোগ্রামের চিত্রভিত্তিক উপস্থাপন।

৯.১৩ অনুশীলনী (Exercises)

অনুশীলনী ৯.১:

১। নিচের প্রোগ্রামের stack diagram আঁক, যা main এবং riddle এর local variables ও parameters দেখাবে এবং এমন objects প্রর্দশন করো যা ভ্যারিয়েবলকে refer করে।

২। এই প্রোগ্রামের আউটপুট কি?

```
public static void main(String[] args)
{
        int x = 5;
        Point blank = new Point(1, 2);
        System.out.println(riddle(x, blank));
        System.out.println(x);
        System.out.println(blank.x);
        System.out.println(blank.y);
}

public static int riddle(int x, Point p)
{
        x = x + 7;
        return x + p.x + p.y;
}
```

এই অনুশীলনীর লক্ষ্য হলো তোমরা প্যারামিটার হিসেবে অবজেক্টকে ব্যবহার করার কৌশলটি বুঝতে পারছো কিনা।

অনুশীলনী ৯.২:

১। নিচের প্রোগ্রামের জন্য, একটি stack diagram আঁকতে হবে যা distance returns করার ঠিক আগেই state of the program দেখাবে। এর মধ্যে সংযুক্ত থাকবে variables ও parameters এবং ভ্যারিয়েবলকে refer করার অবজেক্ট।

২। এই প্রোগ্রামের আউটপুট কি?

```
public static double distance(Point p1, Point p2) {
        int dx = p1.x - p2.x;
        int dy = p1.y - p2.y;
        return Math.sqrt(dx*dx + dy*dy);
}

public static Point findCenter(Rectangle box) {
```

```
        int x = box.x + box.width/2;
        int y = box.y + box.height/2;
        return new Point(x, y);
    }

    public static void main(String[] args) {
        Point blank = new Point(5, 8);

        Rectangle rect = new Rectangle(0, 2, 4, 4);
        Point center = findCenter(rect);

        double dist = distance(center, blank);

        System.out.println(dist);
    }
```

অনুশীলনী ৯.৩:

Rectangle class এর একটি অংশ হলো method তৈরি করা। এই লিংকের নথিগুলো পড়
http://download.oracle.com/javase/6/docs/api/java/awt/Rectangle.html#grow(int,int)

১। এই প্রোগ্রামের আউটপুট কি?

২। একটি state diagram আঁকো যা main এর পর state of the program দেখাবে। সব local variables এবং অবজেক্টগুলোকেও তারা refer করেছে তাও অন্তর্ভুক্ত কর।

৩। main এর শেষে, p1 এবং p2 কি aliased হয়েছে? যদি হয়ে থাকে তাহলে কেন অথবা যদি হয়ে না থাকে তাহলে কেন নয়?

```
    public static void printPoint(Point p) {
        System.out.println( "(" + p.x +  ", "  + p.y + ")" );
    }

    public static Point findCenter(Rectangle box) {
        int x = box.x + box.width/2;
        int y = box.y + box.height/2;
        return new Point(x, y);
    }
    public static void main(String[] args) {

        Rectangle box1 = new Rectangle(2, 4, 7, 9);
        Point p1 = findCenter(box1);
        printPoint(p1);
```

```
        box1.grow(1, 1);
        Point p2 = findCenter(box1);
        printPoint(p2);
    }
```

অনুশীলনী ৯.৪:

এখন হয়তো তোমরা factorial method এর ওপর বিরক্ত হতে পারো, কিন্তু আমার আর একটি ভার্সন করতে যাচ্ছি।

১। Big.java নামে একটি নতুন প্রোগ্রাম তৈরি করো এবং factorial এর একটি পুনরাবৃত্ত (iterative) ভার্সন।

২। 0 থেকে 30 integers এর, তাদের factorials সহ একটি টেবিলে প্রিন্ট করো। 15 এর কাছাকাছি কোন পয়েন্টে তোমরা দেখতে পাবে ফলাফল সঠিক নয়, কেন সঠিক নয় বলো?

৩। BigIntegers হলো জাভা অবজেক্ট যা arbitrarily বড় integers উপস্থাপন করে। এখানে সীমিত memory size এবং processing speed ছাড়া আর কোন upper bound নেই। BigIntegers এর সম্পর্কিত নথিগুলো পড়ার জন্য http://download.oracle.com/javase/6/docs/api/java/math/BigInteger.html এই লিংকে যাও।

৪। BigIntegers ব্যবহার করার জন্য, আমাদের প্রোগ্রামের শুরুতেই import java.math.BigInteger সংযুক্ত করতে হবে।

৫। BigInteger তৈরি করার অনেক উপায় আছে, কিন্তু আমরা বলবো valueOf ব্যবহার করার জন্য। নিচের কোডগুলো integer কে BigInteger পরিবর্তন (converts) করতে পারবে:

```
int x = 17;
BigInteger big = BigInteger.valueOf(x);
```

এই কোডগুলো টাইপ করো এবং চেষ্টা করো। BigInteger প্রিন্ট করার চেষ্টা করো।

৬। যেহেতু BigIntegers, primitive types নয় তাই সাধারণ math operators এখানে কাজ করবে না। এর পরিবর্তে আমাদের add এর মতো methods গুলো ব্যবহার করতে হবে। দুটি BigIntegers যুক্ত করতে আমাদের add on invoke করতে হবে এবং অন্যান্যগুলো আর্গুমেন্ট হিসেবে pass করতে হবে। উদাহরণস্বরূপ:

```
BigInteger small = BigInteger.valueOf(17);
BigInteger big = BigInteger.valueOf(1700000000);
BigInteger total = small.add(big);
```

multiply এবং pow এর মতো অন্যান্য method গুলো ব্যবহারের চেষ্টা করো।

৭। factorial কে পরিবর্তন (Convert) করো যাতে এটি BigIntegers ব্যবহার করে এর হিসেব কষতে পারে। এবং ফলাফল হিসেবে একটি BigInteger রিটার্ন করতে পারে। তুমি এই প্যারামিটারকে একাই ছেড়ে দিতে পারো।

কারণ এটি তখন পর্যন্ত একটি integer।

৮। তোমার পরিবর্তিত factorial method ব্যবহার করে পূর্বের টেবিলটি প্রিন্ট করার জন্য পুনরায় চেষ্টা করো। এটি কি 30 পর্যন্ত সঠিক আছে? কীভাবে আমরা একে সর্বোচ্চ মান পর্যন্ত সঠিক করতে পারি? আমি 0 থেকে 999 পর্যন্ত সকল নম্বরের factorial গণনা করতে পারি কিন্তু আমার যন্ত্র কিছুটা ধীর গতির, তাই এটি কিছুটা সময় নিচ্ছে। শেষ নম্বর হলো 999!, এর ফলাফল হলো 2565 ডিজিটের।

অনুশীলনী ৯.৫:

অনেক encryption কৌশল নির্ভর করে integers এর ওপর integer power কতটা বড় তার ওপর। এখানে একটি method আছে যা একটি হলো দ্রুততার সাথে integer exponentiation বাস্তবায়নের উপায়:

```
public static int pow(int x, int n) {
        if (n == 0) return 1;

        // find x to the n/2 recursively

        int t = pow(x, n/2);

        // if n is even, the result is t squared
        // if n is odd, the result is t squared times x

        if (n%2 == 0) {
                return t*t;
        } else {
                return t*t*x;
        }
}
```

এই method এর সমস্যা হলো এটি তখনই কাজ করবে যখন ফলাফল 2 billion এর ছোট হবে। নতুন করে এটি লিখো যেন এর ফলাফল হয় BigInteger। প্যারামিটার integers-ই হবে কারণ তুমি BigInteger methods; add এবং multiply ব্যবহার করছো কিন্তু তুমি pow ব্যবহার করছো না যা আসল মজাই নষ্ট করে দিবে।

অনুশীলনী ৯.৬:

যদি তুমি graphics এ আগ্রহী হয়ে থাকো তাহলে এটিই Appendix A পড়ার এবং exercises গুলো এখানে করার ভালো সময়।

অধ্যায় ১০

গ্রিড ওয়ার্ল্ড: পর্ব ২ (GridWorld: Part 2)

গ্রিড ওয়ার্ল্ড এর দ্বিতীয় পর্বে কিছু বৈশিষ্ট্য নিয়ে আলোচনা করা হয়েছে যা পূর্বে করা হয় নি, সুতরাং তোমরা এখন একটি প্রাকদর্শন পাবে এবং পরবর্তীতে বিস্তারিত জানতে পারবে। মনে করিয়ে দিচ্ছি, তুমি চাইলে গ্রিড ওয়ার্ল্ড ক্লাসের ডকুমেন্টেশন পড়তে পারবে এখান থেকে:
http://www.greenteapress.com/thinkapjava/javadoc/gridworld/

যখন তুমি গ্রিড ওয়ার্ল্ড ইনস্টল করবে, তোমার একটি ফোল্ডার থাকবে যার নাম projects/boxBug, যার ভেতর থাকবে BoxBugRunner.java এবং BoxBug.gif.

এই ফাইলসমূহ তোমার কাজের ফোল্ডারে অনুলিপি কর এবং ডেভেলপমেন্ট এনভায়রনমেন্ট দ্বারা ইমপোর্ট কর। এই সম্পর্কে নির্দেশনা দেয়া রয়েছে এখানে:
http://www.collegeboard.com/prod_downloads/student/testing/ap/compsci_a/ap07_gridworld_installation_guide.pdf

BoxBugRunner.java এর কোড দেয়া হল:

```
import info.gridworld.actor.ActorWorld;
import info.gridworld.grid.Location;

import java.awt.Color;

public class BoxBugRunner {
        public static void main(String[] args) {
                ActorWorld world = new ActorWorld();
                BoxBug alice = new BoxBug(6);

        alice.setColor(Color.ORANGE);
        BoxBug bob = new BoxBug(3);
        world.add(new Location(7, 8), alice);
        world.add(new Location(5, 5), bob);
        world.show();
        }
}
```

এখানের সবকিছুই পরিচিত লাগার কথা, সম্ভবত Location ছাড়া, এটি হল গ্রিড ওয়ার্ল্ড এর একটি অংশ, এবং java.awt.Point এর অনুরূপ।

BoxBug.java BoxBug এর ক্লাস ডেফিনিশন ধারণ করে।

```
public class BoxBug extends Bug {
        private int steps;
        private int sideLength;

        public BoxBug(int length) {
                steps = 0;
                sideLength = length;
        }
}
```

প্রথম লাইন বলছে এই ক্লাস Bug কে এক্সটেন্ড করে, এর মানে হল BoxBug এক ধরনের Bug.

পরবর্তী দুই লাইন হচ্ছে instance (ঘটনার) ভ্যারিয়েবল। প্রতিটি Bug এর SideLength নামক ভ্যারিয়েবল রয়েছে, যা নির্ধারণ করে অঙ্কিত বক্সের আকার, এবং পদক্ষেপ, যা Bug কতগুলো পদক্ষেপ নিয়েছে তার হিসেব রাখে।

পরবর্তী লাইন একটি constructor এ নির্ধারণ করে, এটি একটি বিশেষ পদ্ধতি যা Instance ভ্যারিয়েবলের সূচনা করে। যখন তুমি একটি Bug তৈরি করার জন্য ডাক, তখন জাভা এই constructor কে ডাকে।

constructor এর প্যারামিটার হচ্ছে SideLength.

Bug এর ব্যবহার নিয়ন্ত্রিত হয় act নামক পদ্ধতি দ্বারা। BoxBug এর act পদ্ধতিটি হল:

```
public void act() {
        if (steps < sideLength && canMove()) {
                move();
                steps++;
        }
                else {
                        turn();
                        turn();
                        steps = 0;
                }
        }
```

যদি BoxBug নড়াচড়া করতে পারে, এবং যদি প্রয়োজনীয় সংখ্যক ধাপ ব্যবহার না করে থাকে, তখন এটি নড়াচড়া করবে এবং ধাপ সংখ্যা বৃদ্ধি করবে।

যদি এটি দেয়ালে ধাক্কা খায় অথবা বাক্সের যেকোন এক পাশ পুরো ঘুরে আসে, তখন এটি ৯০ ডিগ্রী ডান দিকে ঘুরে যাবে এবং ধাপ ০ তে পুনঃনির্ধারণ করবে।

প্রোগ্রামটি চালনা কর এবং দেখ এটি কি করে। তুমি যা চাচ্ছিলে তা কি হচ্ছে?

১০.১ উইপোকা (Termites)

আমি Termite নামক একটি ক্লাস লিখেছি যা Bug কে এক্সটেন্ড করে এবং ফুলের সাথে মিথস্ক্রিয়া ঘটানোর সামর্থ্য অর্জন করে।
এটা চালানোর জন্য নিচের ফাইলসমূহ ডাউনলোড কর এবং তোমার ডেভেলপমেন্ট এনভায়রনমেন্টে ইমপোর্ট কর:

http://thinkapjava.com/code/Termite.java
http://thinkapjava.com/code/Termite.gif
http://thinkapjava.com/code/TermiteRunner.java

যেহেতু Termite এক্সটেন্ড করে Bug কে, তাই Bug এর সকল পদ্ধতি Termite এও কাজ করবে। কিন্তু Termites এর কিছু অতিরিক্ত পদ্ধতি (methods) রয়েছে যা Bug এর নেই।

```
/**
* Returns true if the termite has a flower.
*/
public boolean hasFlower();

/**
* Returns true if the termite is facing a flower.

*/
public boolean seeFlower();

/**
* Creates a flower unless the termite already has one.
*/
public void createFlower();

/**
* Drops the flower in the termites current location.
*
* Note: only one Actor can occupy a grid cell, so the effect
* of dropping a flower is delayed until the termite moves.
*/
public void dropFlower();

/**
* Throws the flower into the location the termite is facing.
*/
public void throwFlower();

/**
```

```
* Picks up the flower the termite is facing, if there is
* one, and if the termite doesn't already have a flower.
*/
public void pickUpFlower() ;
```

কিছু পদ্ধতির জন্য Bug একটি ডেফিনিশন প্রদান করে এবং Termite প্রদান করে অন্য একটি। এই ক্ষেত্রে Termite এর পদ্ধতি Bug এর পদ্ধতিকে বাতিল করে দেয় (override).

উদাহরণস্বরূপ: যদি পরবর্তী অবস্থানে flower থাকে তাহলে Bug.canMove রিটার্ন করে true, তাই বাগ ফুল নষ্ট করতে পারে। যদি পরবর্তী অবস্থানে যেকোন object থাকে তাহলে Termite.canMove রিটার্ন করে false, তাই Termite এর ব্যবহার ভিন্ন হয়।

অন্য একটি উদাহরণ, Termite এর turn এর একটি সংস্করণ রয়েছে যা ডিগ্রী এর পূর্ণ (integer) সংখ্যা প্যারামিটার হিসেবে গ্রহণ করে। সবশেষে Termite এর রয়েছে randomTurn, যার দ্বারা সে ৪৫ ডিগ্রী বামে অথবা ডানে এলোমেলোভাবে ঘুরতে পারে।

এখানে TermiteRunner.java এর কোড দেয়া হল:

```
public class TermiteRunner
{
    public static void main(String[] args)
    {
        ActorWorld world = new ActorWorld();
        makeFlowers(world, 20);
        Termite alice = new Termite();
        world.add(alice);
        Termite bob = new Termite();
        bob.setColor(Color.blue);
        world.add(bob);
        world.show();
    }
    public static void makeFlowers(ActorWorld world, int n) {
        for (int i = 0; i<n; i++) {
            world.add(new EternalFlower());
        }
    }
}
```

এখানের সবকিছুই পরিচিত লাগার কথা। TermiteRunner ২০ টি EternalFlower এবং দুটি Termite দ্বারা একটি ActorWorld তৈরি করে।

একটি EternalFlower হচ্ছে এমন একধরনের ফুল যা act কে বাতিল করে ব্যবহৃত হয় যার কারণে ফুল ফ্যাকাসে হয়ে যায় না।

```
class EternalFlower extends Flower {
        public void act() {}
}
```

যদি তুমি TermiteRunner.java চালাও তাহলে দেখতে পাবে দুটো termite এলোমেলোভাবে ফুলের উপর দিয়ে চলাফেরা করছে।

MyTermite.java ফুলের সাথে যে সকল পদ্ধতি মিথস্ক্রিয়া ঘটায় তা প্রদর্শন করে। এখানে ক্লাস এর ডেফিনিশন দেয়া হল:

```
        public class MyTermite extends Termite {
        public void act() {
                if (getGrid() == null)
                        return;
                if (seeFlower()) {
                        pickUpFlower();
                }
                if (hasFlower()) {
                        dropFlower();
                }
                if (canMove()) {
                move();
                }
                randomTurn();
        }
}
```

MyTermite এক্সটেন্ড করে Termite কে এবং act বাতিল করে নিজেই ব্যবহৃত হয়। যদি MyTermite একটি ফুল দেখতে পায়, তুলে নেয়। যদি ফুল থাকে, তাহলে ফেলে দেয়।

১০.২ ল্যাঙটনের টারমাইট (Langton's Termite)

Langton's Ant হচ্ছে পিঁপড়ার চারিত্রিক বৈশিষ্ট্যের একটি সাধারণ মডেল যা আশ্চর্যজনক জটিল ব্যবহার প্রদর্শন করে। Ant বাস করে একটি GridWorld এর মত একটি গ্রিডে যেখানে প্রতিটি ঘর হয় কালো নয়ত সাদা। পিঁপড়াগুলো নড়াচড়া করে নিচের নিয়ম অনুসারে:

- যদি পিঁপড়া সাদা ঘরে থাকে, এটা ডান দিকে ঘুরে, ঘরটিকে কালো করে এবং সামনের দিকে সরে যায়।
- যদি পিঁপড়া কালো ঘরে থাকে, এটা বাম দিকে ঘুরে, ঘরটিকে সাদা করে এবং সামনের দিকে সরে যায়।

যেহেতু নিয়মগুলো খুবই সাদামাটা, তুমি হয়ত আশা করতে পার পিঁপড়া সাধারণ কিছু করবে যেমন, একটি বর্গক্ষেত্র তৈরি করা অথবা একটি ছোট প্যাটার্ন এর পুনরাবৃত্তি করা। কিন্তু সাদা ঘরে ১০৪ পদক্ষেপের পুনরাবৃত্তির লুপে সেট হওয়ার পূর্বে পিঁপড়া ১০,০০০ এরও অধিক পদক্ষেপ নেয় যার প্যাটার্ন এলোমেলো।

তুমি ল্যাংটনের পিঁপড়া সম্পর্কে আরও জানতে পারবে এখানে http://en.wikipedia.org/wiki/Langton_ant

গ্রিডওয়ার্ল্ডে ল্যাংটনের পিঁপড়া বাস্তবায়ন করা সহজ কাজ নয় কারণ আমরা ঘরের রঙ নির্ধারণ করতে পারি না। এর বদলে, আমরা ঘর চিহ্নিত করার জন্য ফুল ব্যবহার করতে পারি, কিন্তু আমরা একটি ঘরে ফুল এবং পিঁপড়া নিতে পারব না, সুতরাং আমরা যথাযথ ভাবে Ant Rule বাস্তবায়ন করতে পারবে না।

পরিবর্তে আমরা ব্যবহার করব LangtonTermite, যা seeFlower ব্যবহার করে পরীক্ষা করে পরবর্তী ঘরে ফুল রয়েছে কিনা, pickUpFlower ফুল তুলে নেয়, এবং throwFlower পরবর্তী ঘরে ফুলটি রাখে। তুমি যদি নিশ্চিতভাবে জানতে চাও তারা কি করে তাহলে সম্ভবত তোমার এই মেথডের কোড পরার প্রয়োজন হতে পারে।

১০.৩ অনুশীলনী

অনুশীলনী ১০.১: এখন তুমি স্টুডেন্ট ম্যানুয়াল, পর্ব ২ এর অনুশীলনী করার জন্য যথেষ্ট জানো। ওখানে গিয়ে অনুশীলন কর, এবং আরও মজার জন্য ফিরে আস।

অনুশীলনী ১০.২: এই অনুশীলনীর উদ্দেশ্য হচ্ছে ফুলের সাথে মিথস্ক্রিয়া ঘটানো টারমাইটের ব্যবহার অন্বেষণ করা।

TermiteRunner.java পরিবর্তন করে Termites এর পরিবর্তে MyTermites তৈরি কর। তারপর এটি পুনরায় চালু কর। MyTermite এলোমেলোভাবে নড়াচড়া করবে, এবং ফুলগুলোকে নাড়াবে। ফুলের সর্বমোট সংখ্যা একই থাকবে (MyTermite যে ফুলটি ধরে রেখেছে সেটি সহকারে)।

Termites, Turtles and Traffic Jam এ, মাইকেল রেসনিক টারমাইটের ব্যবহারের একটি সহজ মডেল ব্যাখ্যা করেছেন:

- তুমি যদি একটি ফুল দেখ, তুলে নাও। যদি না তোমার কাছে ইতোমধ্যে একটি ফুল থাকে; এক্ষেত্রে যেটি তোমার কাছে রয়েছে সেটি ফেলে দাও।
- সামনে এগিয়ে যাও, যদি সম্ভব হয়।
- বামে অথবা ডানে ঘুরে যাও।

এই মডেল বাস্তবায়নের জন্য MyTermite.java পরিবর্তন কর। MyTermite এ এর ব্যবহারে কি ধরনের পরিবর্তন হবে বলে তুমি মনে কর?

চেষ্টা করে দেখ। এখানেও ফুলের সর্বমোট সংখ্যা পরিবর্তিত হয়নি, কিন্তু সময়ের সাথে সাথে ফুল স্তূপাকারে জমা হয়, প্রায়শই একটি স্তূপ।

এই ব্যবহারটি হচ্ছে একটি উত্থানহীন সম্পত্তি **(an emergent property)**, যা সম্পর্কে তুমি এখানে পড়তে পার http://en.wikipedia.org/wiki/Emergence. MyTermite অল্প তথ্য ব্যবহার করে অনুসরণ করে একটি সাধারণ নিয়ম, কিন্তু এর ফলাফল হয় বৃহদাকারে সংগঠিত।

ভিন্ন নিয়ম নিয়ে পরীক্ষা কর এবং দেখ সিস্টেমে তারা কি ধরনের পরিবর্তন করে। ছোট পরিবর্তন একটি অনিশ্চিত ফলাফল নিয়ে আসতে পারে!

অনুশীলনী ১০.৩:

১। Termite.java এর একটি অনুলিপি কর এবং নাম দাও LangtonTermite.java এবং TermiteRunner.java এর অনুলিপি করে নাম দাও LangtonRunner.java. এমনভাবে পরিবর্তন কর যাতে ক্লাস ডেফিনিশনে এবং ফাইলে একই নাম থাকে, এবং যেন LangtonRunner তৈরি করে LangtonTermite কে।

২। তুমি যদি LangtonTermite.gif নামক একটি ফাইল তৈরি কর, গ্রিডওয়ার্ল্ড এটিকে Termite উপস্থাপনের জন্য ব্যবহার করবে। তুমি কীটপতঙ্গের অসাধারণ ছবি ডাউনলোড করে নিতে পার এখান থেকে http://www.cksinfo.com/animals/insects/realisticdrawings/index.html. এদেরকে GIF ফরমেটে কনভার্ট করার জন্য ImageMagick ব্যবহার করতে পার।

৩। Langton's Ant এর মত নিয়ম বাস্তবায়নের জন্য একটি act পরিবর্তন কর। 45 ও 90 ডিগ্রী টার্ন সহকারে, বিভিন্ন নিয়ম দ্বারা পরীক্ষা কর। এমন একটি নিয়ম বের কর যা টারমাইট লুপে পড়ার পূর্বে অধিক সংখ্যক পদক্ষেপ নিবে।

৪। তোমার টারমাইটকে যথেষ্ট জায়গা দেয়ার জন্য, তুমি গ্রিড বড় করতে পার অথবা UnboundedGrid এ পরিবর্তন করতে পার।

৫। একাধিক LangtonTermite তৈরি কর এবং দেখ তারা কি করে।

অধ্যায় ১১
নিজের অবজেক্ট তৈরী

১১.১ ক্লাসের ডেফিনেশন এবং অবজেক্টের ধরন (Class definitions and object types)

সেকশন ১.৫ এ আমরা যখন একটি ক্লাস hello ডিফাইন করেছিলাম তখন আমরা একটি অবজেক্ট টাইপ তৈরি করে ছিলাম, যার নাম ছিল hello। আমরা hello টাইপ ছাড়া কোন ভ্যারিয়েবল তৈরি করি নাই এবং আমরা কোন hello অবজেক্ট ছাড়া new ব্যবহার করি নাই, কিন্তু আমরা এখন করবো!

এই উদাহরণে খুব বেশি ধারণা দেয় না, যদিও এখানে hello অবজেক্ট তৈরির কোন কারণ নেই, এবং আমরা যদি এটি করি তবে তা খুব বেশি কিছু হবে না। এই অধ্যায়ে, আমরা ক্লাস ডেফিনেশন সম্পর্কে দেখবো যেটা useful অবজেক্ট টাইপ তৈরি করে।

এখানে এই অধ্যায়ের খুব গুরুত্বপূর্ণ ধারণাগুলো দেওয়া হল:

- একটি নতুন ক্লাস ডিফাইন করলে তা সর্বদা একই নামের নতুন অবজেক্ট টাইপ তৈরি করে।
- একটি ক্লাস ডেফিনেশন একটি অবজেক্টের জন্য টেমপ্লেটের মতো: এটা নিরূপণ করে অবজেক্টের কি ভ্যারিয়েবল আছে এবং কোন মেথডে তা পরিচালনা করে।
- প্রত্যেকটি অবজেক্ট কিছু অবজেক্ট টাইপ ধারণ করে; এর অর্থ, এটা কিছু ক্লাসের কিছু ইনসট্যান্স।

- যখন তুমি নতুন অবজেক্ট তৈরির জন্য new যুক্ত করবে, জাভা একটি স্পেশাল মেথড যুক্ত করবে যার নাম constructor এটা করবে প্রাথমিকভাবে ভ্যারিয়েবলের জন্য। তুমি এক বা একাধিক নির্মাতাকে যোগান করতে পারবে একটি ক্লাস ডেফিনেশনের অংশ হিসাবে।

- এমন মেথড যা একটি উপায়ে অপারেট করে তা সেই মেথডের ক্লাস ডেফিনিশনে ডিফাইনকৃত।

এখানে ক্লাস ডেফিনেশনের জন্য কিছু সিনটেক্স ইস্যু দেওয়া হল:

- ক্লাসের নাম (এবং এখন থেকে অবজেক্ট টাইপ) একটি ক্যাপিটাল লেটার দিয়ে শুরু হবে, যা প্রিমিটিভ টাইপ এবং ভ্যারিয়েবল নামের সাথে পার্থক্য বুঝতে সহায়তা করে।

- তুমি সাধারণত প্রতিটি ফাইলে একটি ক্লাস ডেফিনেশন ব্যবহার করবে, এবং অবশ্যই ফাইলের নাম এবং ক্লাসের নাম একই হবে। যেটার শেষে .java থাকবে। উদাহরণস্বরূপ, ফাইলে Time ক্লাসটি ডিফাইন করা হয় এভাবে, Time.java।

- যেকোন প্রোগ্রামে, একটি startup class এ একটি ক্লাস ডিজাইন করা হয়। startup class একটি মেথড ধারণ করে যেটার নাম main, যেটা প্রোগ্রামের শুরুতেই থাকে। আরেকটি ক্লাস may এর একটি মেথড আছে যার নাম main, কিন্তু এটি নির্বাহ হয় না।

এই ইস্যুগুলো দেখতে, চলো আমরা একটি ক্লাস Time দিয়ে একটি উদাহরণ দেখি।

১১.২ টাইম (Time)

একটি অবজেক্ট টাইপ তৈরির সাধারণ মোটিভেশন হলো সম্পর্কযুক্ত ডাটার এনক্যাপসুলেট করা যেটা একটি সিঙ্গেল ইউনিটকে ব্যবহার করে। আমরা ইতিমধ্যে দুইটি টাইপ দেখেছি, Point এবং Rectangle।

আরেকটি উদাহরণ, যেখানে আমরা নিজ থেকে Time কে বাস্তবায়িত করেছি, যেটা এই দিনকে প্রতিনিধিত্ব করে। ডাটা Time অবজেক্টকে ঘন্টা, মিনিট এবং সেকেন্ডে এনক্যাপসুলেট করে। কারণ প্রতিটি Time অবজেক্ট এই ডাটাকে ধারণ করে, আমাদের এইগুলো ধারণ করতে এই ভ্যারিয়েবল দরকার।

প্রথম ধাপ নির্ধারণ করে কোন ধরনের ডাটা প্রতিটি ভ্যারিয়েবলে থাকবে। এটা পরিষ্কার ধারণা দেয় যে, hour, minute অবশ্যই integers। শুধুমাত্র আগ্রহ ধরে রাখতে, second কে doouble এ তৈরি করে দেখা যেতে পারে।

ইনট্যান্সস ভ্যারিয়েবল ক্লাস ডেফিনেশনের প্রথমেই ডিক্লেয়ার করা হয়, এটি করা হয় যেকোন মেথড ডেফিনেশনের বাইরে, যেমন:

```
class Time {
        int hour, minute;
        double second;
}
```

এই কোড অংশটুকু নিজস্বভাবেই একটি বৈধ ক্লাস ডেফিনেশন। Time অবজেক্ট এর স্টেট ডায়াগ্রাম অনেকটা এই

রকম:

ইনসট্যান্স ভ্যারিয়েবল ডিক্লেয়ারের পর, পরবর্তী ধাপ হল নতুন ক্লাসের জন্য একটি কনসট্রাক্টর ডিফাইন করা।

১১.৩ কনসট্রাক্টর (Constructors)

কনসট্রাক্টর ইনসট্যান্স ভ্যারিয়েবল আরম্ভ করে। তিনটি ব্যতিক্রমসহ কনসট্রাক্টর এর জন্য সিনটেক্স অন্যান্য মেথডের মতোই:

- কন্সট্রাকটরের নাম এবং ক্লাসের নাম একই হবে।
- কন্সট্রাকটরের কোন রিটার্ন টাইপ এবং রিটার্ন ভ্যালু থাকে না।
- static কীওয়ার্ড বাদ দেওয়া হয়।

Time ক্লাসের একটি উদাহরণ দেওয়া হল:

```
public Time() {
        this.hour = 0;
        this.minute = 0;
        this.second = 0.0;
}
```

তুমি কোথায় একটি রিটার্ন টাইপ দেখতে চাও, public এবং Time এর মাঝে কিছুই নেই। এই কারণে আমরা (এবং কম্পাইলার) বলতে পারে যে এটি একটি কনসট্রাক্টর।

এই কনসট্রাক্টর কোন আর্গুমেন্ট গ্রহণ করে না। কন্সট্রাকটরের প্রতিটি লাইন ডিফল্ট ভ্যালু হিসাবে ইনসট্যান্স ভ্যারিয়েবল আরম্ভ করে (এই ক্ষেত্রে, midnight)। this একটি বিশেষ কীওয়ার্ড যেটা অবজেক্ট উল্লেখ করে এবং এটি আমরা তৈরি করেছি। তুমি এটি একই উপায়ে ব্যবহার করতে পার, যেভাবে তুমি একটি অবজেক্ট এর অন্য নাম দাও সেভাবে। উদাহরণস্বরূপ, তুমি this এর ইনসট্যান্স ভ্যারিয়েবলের রিড এবং রাইট করতে পার, এবং তুমি এটি অন্য একটি মেথডের আর্গুমেন্ট হিসাবে পাস করতে পার।

কিন্তু তুমি this ডিক্লেয়ার করতে পারবে না এবং তুমি এটার কোন অ্যাসাইনমেন্ট তৈরি করতে পারবে না। এটি সিস্টেম দ্বারা তৈরি করা হবে; এর সব তোমার করতে হবে ইনসট্যান্স ভ্যারিয়েবল আরম্ভ হওয়ার সময় থেকেই।

একটি কমন এরর হল, যখন কনসট্রাক্টর রাইট করা হয় তখন শেষে একটি return স্টেটমেন্ট রাখা। তাই এক্ষেত্রে তাড়াহুড়া করা যাবে না।

১১.৪ আরও কনসট্রাক্টর (More constructors)

অন্যান্য মেথডের মতো কনসট্রাক্টর ওভারলোডেড হতে পারে, যার অর্থ দাঁড়ায় যে, তুমি বিভিন্ন প্যারামিটারের সাথে একাধিক কনসট্রাক্টর প্রোভাইড করতে পারবে। জাভা জানে new এর আর্গুমেন্টের সাথে কনসট্রাক্টর এর প্যারামিটার match করে কোন কনসট্রাক্টর ইনভোক করতে হবে।

এটা সাধারণ যে একটি কনসট্রাক্টর থাকে যেটা কোন আর্গুমেন্ট গ্রহণ করে না (উপরে প্রদর্শিত), এবং একটি কনসট্রাক্টর থাকে যেটা ইনসট্যান্স ভ্যারিয়েবলের লিস্ট থেকে একটি প্যারামিটার লিস্ট গ্রহণ করে। উদাহরণস্বরূপ:

```
public Time(int hour, int minute, double second) {
        this.hour = hour;
        this.minute = minute;
        this.second = second;
}
```

প্যারামিটারের name এবং type আর ইনসট্যান্স ভ্যারিয়েবলের name এবং type একই। প্রতিটি কনসট্রাক্টর ইনসট্যান্স ভ্যারিয়েবলের প্যারামিটার থেকে এর তথ্য কপি করে।

তুমি যদি points এবং Rectangle এর ডকুমেন্টেশন দেখ, তবে তুমি দেখবে যে দুই ক্লাসই এই রকম কনসট্রাক্টর প্রোভাইড করে। প্রথমে ওভারলোডিং কনসট্রাক্টর একটি অবজেক্ট তৈরিতে নমনীয়তা প্রদান করে এবং তারপর শূন্যস্থান পূরণ করে, অথবা অবজেক্ট তৈরির আগের সব তথ্য সংগ্রহ করে।

এটা খুব একটা আকর্ষণীয় মনে নাও হতে পারে, এবং হয়তো বা এটা নাও হতে পারে। কনসট্রাক্টর একটি বিরক্তিকর এবং যান্ত্রিক প্রক্রিয়া। একবার যদি তুমি দুইটা রাইট করতে পার, তবে দেখবে যে, তুমি খুব দ্রুত ইনসট্যান্স ভ্যারিয়েবলের তালিকা দেখেই এগুলো রাইট করতে পারছ।

১১.৫ নতুন অবজেক্ট তৈরি

যদিও কনস্ট্রাকটর মেথডের মতোই, কিন্তু তুমি কখনও এদের সরাসরি যুক্ত করবে না। এর পরিবর্তে, যখন তুমি new যুক্ত করবে, সিস্টেম নতুন অবজেক্টের জন্য খালি জায়গা ভাগ করে দেয়, এবং এরপর তোমার কনস্ট্রাকটর যুক্ত করে।

নিচের প্রোগ্রামটি Time অবজেক্টটি তৈরি এবং আরম্ভ করার দুইটি উপায় প্রদর্শিত করে:

```
class Time {
        int hour, minute;
        double second;

        public Time() {
                this.hour = 0;
                this.minute = 0;
                this.second = 0.0;
        }
```

```
	public Time(int  hour, int minute, double second) {
		this.hour = hour;
		this.minute = minute;
		this.second = second;
	}

	public static void main(String[] args) {

		// one way to create and initialize a Time object
		Time t1 = new Time();
		t1.hour = 11;
		t1.minute = 8;
		t1.second = 3.14159;
		System.out.println(t1);

		// another way to do the same thing
		Time t2 = new Time(11, 8, 3.14159);
		System.out.println(t2);
	}
}
```

main এ, প্রথমবার আমরা new যুক্ত করেছিলাম, আমরা নতুন আর্গুমেন্ট প্রদান করেছিলাম, তাই জাভা প্রথম কনস্ট্রাকটর যুক্ত করে। পরের কিছু লাইন ইনসট্যান্স ভ্যারিয়েবলের মান নির্দেশ করে।

দ্বিতীয় বার আমরা new যুক্ত করেছিলাম, আমরা আর্গুমেন্ট প্রদান করেছিলাম যা দ্বিতীয় কনস্ট্রাকটরের প্যারামিটারের সাথে মিলে যায়। এই উপায়ে ইনসট্যান্স ভ্যারিয়েবলের আরম্ভ করা আরও বেশি সংক্ষেপিত এবং আরও বেশি কার্যকরী, কিন্তু এটা রিড করা অনেক বেশি কঠিন, অনেক সময় এটা পরিষ্কার না যে, কোন ইনসট্যান্স ভ্যারিয়েবলের জন্য কোন মানটি নির্দেশিত করা হয়েছে।

১১.৬ অবজেক্ট প্রিন্ট করা (Printing objects)

এই প্রোগ্রামের আউটপুট হল:

```
Time@80cc7c0
Time@80cc807
```

যখন জাভা একটি ইউজার-ডিফাইন অবজেক্ট টাইপের ভ্যালু প্রিন্ট করে, তখন এটি টাইপের নাম প্রিন্ট করে এবং একটি বিশেষ হেক্সাডিসিমল (১৬ ভিত্তিক) কোড প্রিন্ট করে যেটা অন্য অবজেক্ট থেকে অনন্য। এই কোডের নিজের কাছে কোন অর্থ নেই; যদিও, এটা মেশিন ভেদে ভিন্ন হতে পারে এমনকি রান থেকে রানেও ভিন্ন হতে পারে। কিন্তু এটি ডিবাগিং এর জন্য উপকারী, যদি তুমি এই আলাদা অবজেক্টের জন্য ট্র্যাক রাখতে চাও।

তুমি যদি ব্যবহারকারীর জন্য অর্থপূর্ণ একটি উপায়ে অবজেক্ট প্রিন্ট করতে চাও (প্রোগ্রামারদের হিসাবে বিরোধিতা), তুমি printTime নামে একটি মেথড কল করতে পার:

```
public static void printTime(Time t) {
        System.out.println(t.hour + ":" + t.minute + ":" + t.second);
}
```

সেকশন ৩.১০ এর printTime ভার্সনের সাথে এই মেথডটি তুলনা করতে পার। আমরা যদি t1 অথবা t2 আর্গুমেন্ট হিসাবে পাস করি, তবে এই মেথডের আউটপুট 11:8:3.14159 হবে। যদিও এটি সময় হিসাবে স্বীকৃত, তবে এটা পুরোপুরি আদর্শ ফরমেট না। উদাহরণস্বরূপ, যদি মিনিট অথবা সেকেন্ডের সংখ্যা ১০ এর কম হয়, তাহলে আমরা প্লেস-কিপার হিসাবে একটি শূন্যকে আশা করতে পারি। এছাড়াও, আমরা সেকেন্ডের ডেসিমল অংশটুকু ফেলে দিতে পারি। অন্য কথায়, আমরা এই রকমও করতে পারি যেমন, 11:08:03 .

অধিকাংশ ল্যাঙ্গুয়েজে, আউটপুট সংখ্যার ফরমেট কিরূপ হবে তা নিয়ন্ত্রণের জন্য খুব সহজ কিছু উপায় আছে। কিন্তু জাভায় কোন সহজ উপায় নেই।

প্রিন্টেড ফরমেটেড কিছু যেমন সময় এবং তারিখের জন্য জাভা শক্তিশালী টুল প্রদান করে এবং এটা করা হয় সর্বদা ইনপুট ফরমেটের ব্যাখ্যা করতে। দুর্ভাগ্যক্রমে, এই টুলগুলো ব্যবহার করা খুব একটা সহজ না, তাই আমি এই বইয়ে এই বিষয় নিয়ে আলোচনা করবো না। যদি তুমি আরও জানতে চাও, তবে তুমি java.util প্যাকেজের Data ক্লাসের ডকুমেন্টেশন দেখতে পার।

১১.৭ অবজেক্টের অপারেশন (Operations on objects)

পরবর্তী কয়েকটি সেকশনে, আমি তিন প্রকার মেথডের সাথে পরিচয় করিয়ে দেব যেগুলো একটি অবজেক্ট অপারেট করে:

পিওর ফাংশন (pure fanction): একটি ফাংশনকে আর্গুমেন্ট হিসাবে নেওয়া কিন্তু তাকে সংশোধন না করা। রিটার্ন ভ্যালু হতে পারে একটি আদিম অথবা মেথডের ভেতর তৈরিকৃত নতুন অবজেক্ট।

মডিফায়ার (modifier): অবজেক্টকে আর্গুমেন্ট হিসাবে নেয় এবং এর কিছু অংশ অথবা পুরোটা পরিবর্তন করে। প্রায়ই void রিটার্ন করে।

ফিল ইন মেথড (fill-in method): “empty” অবজেক্ট একটি আর্গুমেন্ট যা পূর্ণ হয় একটি মেথড দিয়ে। প্রযুক্তিগতভাবে, এটি একধরনের মডিফায়ার।

প্রায়শই পিওর ফাংশন, মডিফায়ার অথবা ফিল-ইন মেথড হিসাবে একটি মেথড রাইট করা সম্ভব। আমি এর প্রতিটির সুবিধা অসুবিধা নিয়ে বিস্তারিত লিখব।

১১.৮ পিওর ফাংশন (Pure functions):

একটি মেথড একটি পিওর ফাংশন হিসাবে বিবেচিত হতে পারে যদি ফলাফল শুধুমাত্র আর্গুমেন্টের উপর নির্ভর করে, এবং আর্গুমেন্ট এর কোন পরিবর্তন অথবা কোন কিছু প্রিন্ট করার মতো এর কোন পার্শ্বপ্রতিক্রিয়া থাকে না। একটি পিওর ফাংশন যুক্ত করার একটি মাত্র ফল হল রিটার্ন ভ্যালু।

একটি উদাহরণ হল isAfter, যেখানে দুইটা Time এর তুলনা করা হয়েছে এবং একটি বুলিয়ান রিটার্ন করা হয়েছে যেটা নির্দেশ করে প্রথম অপারেন্ড দ্বিতীয় অপারেন্ডের পরে আসে কিনা:

```
public static boolean isAfter(Time time1, Time time2) {
        if (time1.hour > time2.hour) return true;
        if (time1.hour < time2.hour) return false;

        if (time1.minute > time2.minute) return true;
        if (time1.minute < time2.minute) return false;

        if (time1.second > time2.second) return true;
        return false;
}
```

এই মেথডের ফলাফল কি হবে যদি দুইটি সময়ই (times) সমান হয়? এই মেথডের জন্য এই ফলাফল কি যথাযথ মনে হয়? তুমি যদি এই মেথডের জন্য ডকুমেন্টেশন রাইট কর, তুমি কি তাহলে এই ক্ষেত্রটি আলাদা ভাবে উপস্থাপন করবে?

addTime হল দ্বিতীয় আরেকটি উদাহরণ, যেটা দুইটি সময়ের যোগফল গণনা করে। উদাহরণস্বরূপ, যদি এখন সময় 9:14:30 হয়, এবং তোমার রুটি তৈরির কারিগর এর জন্য ৩ ঘন্টা এবং ৩৫ মিনিট সময় নেয়, তাহলে তুমি কত রুটি তৈরিতে কত সময় লাগবে তা পরিমাপ করতে addTime ব্যবহার করতে পারবে।

এখানে এই মেথডের একটি খসড়া উদাহরণ দেওয়া হল, যেটা পুরোপুরি সঠিক নয়:

```
public static Time addTime(Time t1, Time t2) {
        Time sum = new Time();
        sum.hour = t1.hour + t2.hour;
        sum.minute = t1.minute + t2.minute;
        sum.second = t1.second + t2.second;
        return sum;
}
```

যদিও এই মেথড Time অবজেক্ট রিটার্ন করে, তবে এটি কনসট্রাক্টর না। তুমি চাইলে পূর্বের অধ্যায়ে ফিরে যেতে পার এবং মেথডের সিনটেক্সকে তুলনা কর যেন এটা একটি কন্সট্রাকটরের সিনটেক্সের সাথে থাকে, কারণ এটা খুব সহজেই বিভ্রান্ত করে দেয়।

এই মেথডটি কিভাবে ব্যবহার করতে হবে তার একটি উদাহরণ দেওয়া হল। যদি currentTime বর্তমান সময়কে ধরে রাখে এবং breadTime রুটি তৈরির কারিগরের কত সময় লাগে সেই পরিমিত সময়টা ধরে রাখে, তাহলে তুমি addTime ব্যবহার করতে পার, কখন রুটি তৈরি হবে তার।

```
Time currentTime = new Time(9, 14, 30.0);
Time breadTime = new Time(3, 35, 0.0);
Time doneTime = addTime(currentTime, breadTime);
printTime(doneTime);
```

এই প্রোগ্রামের আউটপুট হল 12:49:30.0 , যেটা সঠিক। অন্য দিকে, অনেকক্ষেত্রে আউটপুট সঠিক হয় না। যে কোন একটি মনে করে দেখ?

সমস্যা হল যে এই মেথড কেসের সাথে কাজ করে না যেখানে সেকেন্ড এবং মিনিট ৬০ এর বেশি যোগ করা যায় না। এই ক্ষেত্রে, আমাদের অতিরিক্ত সেকেন্ড মিনিটের কলামে “carry” করতে হবে অথবা অতিরিক্ত মিনিট ঘন্টার কলামে “carry” করতে হবে।

এখানে এই মেথডের সঠিক ভার্সনটি দেওয়া হল।

```
public static Time addTime(Time t1, Time t2) {
        Time sum = new Time();
        sum.hour = t1.hour + t2.hour;
        sum.minute = t1.minute + t2.minute;
        sum.second = t1.second + t2.second;

        if (sum.second >= 60.0) {
                sum.second -= 60.0;
                sum.minute += 1;
        }
        if (sum.minute >= 60) {
                sum.minute -= 60;
                sum.hour += 1;
        }
        return sum;
}
```

যদিও এটি সঠিক, তবে এটি বড় কিছু থেকে শুরু হয়েছে। পরবর্তীতে আমি এর বিকল্প হিসাবে ছোট কিছুর পরামর্শ দেব।

এই কোড দুইটি অপারেটর += এবং -= এর সাথে আমাদের পরিচয় করে দেয় যা আমরা আগে কখনও দেখিনি। এই অপারেটরগুলো ভ্যারিয়েবলকে কমানো এবং বাড়ানোর সংক্ষিপ্ত উপায়। এগুলো ++ এবং - - এর সমান, ব্যতিক্রম (1) তারা int এর মতো double এর উপর কাজ করে এবং (2) কমে যাওয়ার মান 1 হতে হবে তা নয়। স্টেটমেন্ট sum.second -= 60.0 ; sum.second = sum.second - 60 এর সমান।

১১.৯ মডিফায়ার (Modifiers)

মডিফায়ারের উদাহরণের মতো, increment বিবেচনা করা হল, যেটা সেকেন্ডের Time অবজেক্টে প্রদত্ত সংখ্যা যোগ করে। আবারও এই মেথডের জন্য একটি খসড়া প্রোগ্রাম রচনা করা হল:

```
public static void increment(Time time, double secs) {
        time.second += secs;

        if (time.second >= 60.0) {
                time.second -= 60.0;
                time.minute += 1;
        }
```

```
        if (time.minute >= 60) {
            time.minute -= 60;
            time.hour += 1;
        }
    }
```

প্রথম লাইনটি বেসিক অপারেশন পরিচালনা করে; একই কেসে আমরা অবশিষ্ট অংশটুকু আগেই দেখেছিলাম।

এই মেথড কি ঠিক আছে? যদি আর্গুমেন্ট secs ৬০ এর চেয়ে বড় হয় তখন কি হবে? এই ক্ষেত্রে, একবার ৬০ বিয়োগ করাই যথেষ্ট নয়; আমরা এটা ততক্ষণ করবো যতক্ষণ না second ৬০ এর নিচে হয়। if স্টেটমেন্টের পরিবর্তে while স্টেটমেন্ট ব্যবহার করে আমরা এটি করতে পারি:

```
    public static void increment(Time time, double secs) {
        time.second += secs;

        while (time.second >= 60.0) {
            time.second -= 60.0;
            time.minute += 1;
        }
        while (time.minute >= 60) {
            time.minute -= 60;
            time.hour += 1;
        }
    }
```

এই সমাধানটি সঠিক, কিন্তু খুব বেশি কার্যকর না। তুমি কি এমন একটি কার্যকর সমাধান বের করতে পার যেখানে পুনরাবৃত্তি করার প্রয়োজন পড়েনা?

১১.১০ ফিল-ইন মেথডস (Fill-in methods)

প্রতিবার নতুন অবজেক্ট তৈরি করার পরিবর্তে addTime যুক্ত কর, আমরা কলারকে একটি অবজেক্ট প্রদান করতে বলতে পারি যেখানে addTime ফলাফল সংরক্ষণ করবে। পূর্বের ভার্সনের সাথে এটি তুলনা কর:

```
    public static void addTimeFill(Time t1, Time t2, Time sum) {
        sum.hour = t1.hour + t2.hour;
        sum.minute = t1.minute + t2.minute;
        sum.second = t1.second + t2.second;

        if (sum.second >= 60.0) {
            sum.second -= 60.0;
            sum.minute += 1;
        }
        if (sum.minute >= 60) {
```

```
            sum.minute -= 60;
            sum.hour += 1;
        }
    }
```

রেজাল্টটা sum এ সংরক্ষণ হয়েছে, তাই রিটার্ন টাইপ void হবে।

মডিফায়ার এবং ফিল-ইন মেথড দুইটি কার্যকর কারণ তারা নতুন অবজেক্ট তৈরি করে না। কিন্তু তারা প্রোগ্রামকে বিভিন্ন অংশে ভাগ করে আরও জটিল করে তুলে; বড় প্রোগ্রামে তারা ইররের কারণ হতে পারে যা খুঁজে পাওয়া কঠিন।

পিওর ফাংশন বড় প্রোগ্রামের জটিলতাকে নিয়ন্ত্রণ করতে সহায়তা করে, অংশে বিভক্ত করায় এই জাতীয় নিশ্চিত ভুল হওয়া অসম্ভব। যদিও, তারা নিজেরা নিজেদের ধার করে কিছু নির্দিষ্ট প্রকারের কম্পোজিশন এবং নেস্টিং এর জন্য। এবং কারণ পিওর ফাংশনের ফলাফল নির্ভর করে শুধুমাত্র প্যারামিটারের উপর, পূর্ববর্তী শক্তিশালী গাণিতিক হিসাবকৃত ফলাফলের জন্য তাদের গতি বাড়ানো সম্ভব হয়।

আমি তোমাকে পিওর ফাংশন লিখতে সুপারিশ করব যেটা যুক্তিযুক্ত, এবং মডিফায়ারকে অবলম্বন করতে বলব শুধুমাত্র যদি কম্পাইলারের সুবিধা থাকে।

১১.১১ ইনক্রিমেন্টাল ডেভেলপমেন্ট এবং পরিকল্পনা (Incremental development and planning)

এই সেকশনে আমি প্রোগ্রাম ডেভেলপমেন্টের একটি প্রক্রিয়া দেখাবো যাকে র‍্যাপিড প্রটোটাইপিং[3] বলা হয়। প্রতিটি মেথড, আমি যেগুলো খসড়া হিসাবে লিখেছিলাম সেগুলো বেসিক গাণিতিক হিসাব করতে পারে। এরপর আমি কয়েকটি পরীক্ষা করেছিলাম, আমি যে ত্রুটিগুলো পেয়েছি তা সংশোধন করেছি।

এই পদ্ধতিটি কার্যকর হতে পারে, কিন্তু এটা কোডকে নেতৃত্ব দিতে পারে যেটা অপ্রয়োজনীয়ভাবে জটিল- যেহেতু এটি অনেকগুলো বিশেষ কেসের সাথে কাজ করে- এবং অবিশ্বাস্য- যেহেতু এটা তোমাকে সন্তুষ্ট করতে পারে যে তুমি সকল এরর খুঁজে পেয়েছ।

একটি বিকল্প হল সমস্যার ভেতরে দেখা যেটা প্রোগ্রামিংকে অনেক সহজ করে দেয়। এই ক্ষেত্রে প্রোগ্রামের ভেতর দেখতে পাই Time হল তিন ডিজিটের সংখ্যা, যেখানে ভিত্তি হল 60! সেকেন্ড হয় "ones column" এর, মিনিট "60's column" এবং ঘন্টা হয় "3600's column" এর।

যখন আমরা লিখি addTime এবং increment, তখন আমরা ৬০ কে ভিত্তি ধরে কার্যকরভাবে যোগ করি, এই কারণে আমাদের এক কলাম থেকে আরেক কলামে সংখ্যা নিয়ে যেতে হয়।

সমগ্র সমস্যার আরেকটি এপ্রোচ হল Time এর মধ্যে double কে রূপান্তর করা এবং এর সুবিধা হল কম্পিউটার আগেই জানে যে কিভাবে double এর সাথে গাণিতিক হিসাব করতে হয়। এখানে একটি মেথড দেওয়া হল যেখানে double এর মাঝে Time কে রূপান্তর করা হয়েছে:

---

3 রেপিড প্রটোটাইপিং যেটাকে আমি কল করেছিলাম তা টেস্ট-ড্রাইভেন ডেভেলপমেন্ট (TDD) এর মতই; পার্থক্য হল যে, TDD স্বভাবতই অটোমেটেড টেস্টিং এর উপর নির্ভর করে। বিস্তারিত দেখ এখানে http://en.wikipedia.org/wiki/Test-driven_development

```
        public static double convertToSeconds(Time t) {
        int minutes = t.hour * 60 + t.minute;
        double seconds = minutes * 60 + t.second;
        return seconds;
}
```

এখন আমাদের double থেকে Time অবজেক্টে রূপান্তরের উপায় দরকার। আমরা এর জন্য একটি মেথড রাইট করতে পারি, কিন্তু এটা তৃতীয় একটি কনসট্রাক্টর রাইট করতে খুব বেশি অর্থ তৈরি করে :

```
public Time(double secs) {
        this.hour =(int)(secs / 3600.0);
        secs -= this.hour * 3600.0;
        this.minute =(int)(secs / 60.0);
        secs -= this.minute * 60;
        this.second = secs;
}
```

এই কনসট্রাক্টর অন্যগুলো থেকে আলাদা; এটা ইনসট্যান্স ভ্যারিয়েবলের অ্যাসাইনমেন্টের সাথে কিছু ক্যালকুলেশন যুক্ত করে।

তোমার অবশ্যই নিজেকে সন্তুষ্ট করতে চিন্তা করতে হবে যে, আমি যে টেকনিকটি খাটিয়েছি এক বেস থেকে আরেক বেসে রূপান্তরের জন্য তা ঠিক। কিন্তু তুমি যদি একবার সন্তুষ্ট হও, তুমি এই মেথডগুলো addTime পুনরায় লিখতে ব্যবহার করবে:

```
        public static Time addTime(Time t1, Time t2) {
                double seconds = convertToSeconds(t1) + convertToSeconds(t2);
                return new Time(seconds);
        }
```

এটা অরজিনাল ভার্সন থেকে সংক্ষিপ্ত, এবং এটা যে সঠিক তা প্রদর্শন করা আরও বেশি সহজ (স্বাভাবিক ভাবে, যে মেথডগুলো যুক্ত করা হয়েছে তা সঠিক)। অনুশীলনী হিসাবে, একই ভাবে increment রিরাইট কর।

১১.১২ সর্বজনীনকরণ (Generalization)

বেস 60 থেকে বেস 10 এ কনভার্ট করা শুধুমাত্র সময় নিয়ে কাজ করার চাইতে কঠিন। Base রূপান্তর অনেক বেশি অবাস্তব (abstract); আমাদের time নিয়ে কাজ করার অনুভূতি অনেকটাই ভাল।

যদি আমরা সময়কে বেস 60 সংখ্যা ধরে ব্যবহার করি, এবং কনভার্সেশন মেথডকে রাইট করাতে ইনভেস্টমেন্ট তৈরি করতে পারি (convertToSeconds এবং তৃতীয় কনসট্রাক্টর), আমরা একটি প্রোগ্রাম পাব যা রাইট এবং ডিবাগ করতে অনেক বেশি সংক্ষিপ্ত, সহজ এবং বেশি বিশ্বাসযোগ্য।

এটাতে পরবর্তীতে কোন বৈশিষ্ট্য যোগ করাও সহজ। দুইবার Times যোগ করার কথা ভাব, এদের মাঝে পার্থক্য কতক্ষণের তা খুঁজে বের কর। সাদামাটাভাবে বিয়োগের কাজ সম্পন্ন করতে অনেক সময়ই “ধার (borrowing)”

নিতে হয়। কনভার্সেশন মেথড ব্যবহার করলে তা অনেকটাই সহজ হয়ে যায়।

হাস্যকরভাবে, কোন কোন সময় সমস্যাটি কঠিন হতে পারে (খুবই সাধারণ), এটিকে সহজ কর (খুবই কম বিশেষ কিছু ক্ষেত্রে, এরর এর জন্য খুব কমই সুযোগ থাকে)।

১১.১৩ অ্যালগরিদম (Algorithms)

একটি ক্লাসের সমস্যার জন্য যখন তুমি একটি সর্বজনীন সমাধান রাইট করবে, একটি সিঙ্গেল সমস্যার সুনির্দিষ্ট সমাধানে বিরোধিতা করবে, তখন তোমাকে একটি অ্যালগরিদম (algorithm) রাইট করতে হবে। এই শব্দটি সংজ্ঞায়িত করা সহজ না, তাই আমি তোমাকে কিছু উপায়ে এটি দেখাবো।

প্রথমত, কিছু বিষয় নিয়ে ভাব যা অ্যালগরিদম নয়। যখন তুমি একটি সিঙ্গেল ডিজিট সংখ্যার গুণ করবে, তুমি সম্ভবত নামতার তালিকাকে মনে করবে। এই ফলাফলের জন্য, তুমি প্রায় ১০০টি সুনির্দিষ্ট সমাধান মনে করবে, তাই বলা যায় জ্ঞান সত্যিকারের অ্যালগরিদম নয়।

কিন্তু তুমি যদি "অলস" হও তবে কিছু কৌশল অবলম্বন করবে। উদাহরণস্বরূপ, n এবং 9 এর গুণফল বের করতে, তুমি প্রথম ডিজিট হিসাবে n-1 লিখবে এবং দ্বিতীয় ডিজিট হিসাবে 10-n লিখবে। যেকোন একক সংখ্যাকে 9 দ্বারা গুন করতে এটি একটি সাধারণ কৌশল। এটাই একটা অ্যালগরিদম!

একইভাবে, তুমি হাতে রেখে যোগ করতে, ধার করে বিয়োগ করতে এবং ভাগশেষ পাওয়া পর্যন্ত ভাগ করার যে কৌশল তুমি শিখেছ তার সবই লগারিদম। লগারিদমের একটি বৈশিষ্ট্য হল যে, এগুলো বের করতে কোন বুদ্ধিমত্তার দরকার পরে না। এগুলো মেশিনগত প্রক্রিয়া যেগুলো প্রতিটি ধাপে শেষ ধাপ অনুসারে কিছু পূর্ব নির্ধারিত নিয়ম পালন করে।

আমার মতে, এগুলো খুব লজ্জায় ফেলে যে, মানুষ তার বেশির ভাগ সময়ই এই গুলোর সাহায্যে অ্যালগরিদম করতে বিদ্যালয়ে কাটায়, কমবেশি অক্ষরজ্ঞান লাভ করে, এতে কোন বুদ্ধিমত্তার প্রয়োজন পরে না। অন্য দিকে, অ্যালগরিদমের ডিজাইন প্রক্রিয়া খুবই আকর্ষণীয়, বুদ্ধিবৃত্তিকভাবে প্রতিযোগিতামূলক, এবং আমরা কি ধরনের প্রোগ্রাম করতে যাচ্ছি তার কেন্দ্রীয় অংশ।

কিছু জিনিস আছে যা লোকজন স্বভাবতই করে থাকে, এগুলো করা হয় কঠিনতা অথবা সচেতনভাবে চিন্তা ছাড়াই, সেগুলোই সুস্পষ্টভাবে এলগরিদমে ব্যাখ্যা করা কঠিন হয়ে দাঁড়ায়। ন্যাচারাল ভাষা বোঝা এর একটি ভাল উদাহরণ। আমরা সবাই এটা করি, কিন্তু কেউ একজন ব্যাখ্যা করতে পারবে না আমরা কিভাবে এটা করে থাকি। এমন কি এটা অ্যালগরিদমের কোন ফর্মেও পড়বে না।

খুব শীঘ্রই একটি নানাবিধ সমস্যার জন্য সহজ সরল একটি অ্যালগরিদম ডিজাইন করার সুযোগ তোমার আসবে।

১১.১৪ শব্দকোষ (Glossary)

ক্লাস (class): পূর্বে, আমি একটি ক্লাসকে সম্পর্কযুক্ত মেথডের সংগ্রহ হিসাবে ডিফাইন করেছিলাম। এই অধ্যায়ে আমরা শিখলাম যে, একটি ক্লাস ডেফিনেশন সর্বদাই নতুন ধরনের অবজেক্টের জন্য টেমপ্লেট হয়।

ইনসট্যান্স (instance): একটি ক্লাসের সংখ্যা। প্রতিটি অবজেক্ট কিছু ক্লাসের ইনসট্যান্স হয়ে থাকে।

কনসট্রাক্টর (constructor): একটি বিশেষ মেথড যেটা নতুন তৈরিকৃত অবজেক্টের জন্য ইনসট্যান্স ভ্যারিয়েবল আরম্ভ করে।

স্টার্টআপ ক্লাস (startup class): একটি ক্লাস যেটি যেখানে প্রোগ্রামের এক্সিকিউশন শুরু হয় সেখানে main মেথড ধারণ করে।

পিওর ফাংশন (pure function): একটি মেথড যার ফলাফল নির্ভর করে কেবল মাত্র এর প্যারামিটারের উপর, এবং যখন এটি ভ্যালু রিটার্ন করে তখন এর কোন পার্শ্বপ্রতিক্রিয়া নেই।

মডিফায়ার (modifier): একটি মেথড যা একটি কিংবা দুইটি অবজেক্টকে পরিবর্তন করে এটি প্যারামিটার হিসাবে গৃহীত হয়, এবং প্রচলিতভাবেই void রিটার্ন করে।

ফিল-ইন মেথড (fill-in method): এক ধরনের মেথড যা প্যারামিটার হিসাবে একটি "empty" অবজেক্ট নেয় এবং একটি রিটার্ন ভ্যালু উৎপাদন করার পরিবর্তে এটার ইনসট্যান্স ভ্যারিয়েবল দিয়ে পূর্ণ করে।

অ্যালগরিদম (algorithm): একটি মেকানিক্যাল প্রক্রিয়ায় সমস্যার সমাধান বের করার জন্য ইন্সট্রাকটরের সেট।

১১.১৫ অনুশীলনী

অনুশীলনী ১১.১: Scrabble[4] বোর্গ গেমে, প্রতিটি টালি একটি অক্ষর ধারণ করে, যেটা ব্যবহার করা হয় শব্দ বানান করতে, এবং স্কোর এ, যেটা ব্যবহৃত হয় শব্দের মান নির্ণয়ে।

1. Tile নামক ক্লাসের এর একটি ডেফিনেশন লিখ যেটা Scrabble এর টালিকে উপস্থাপন করে। ইনসট্যান্স ভ্যারিয়েবল অবশ্যই একটি ক্যারেক্টর হবে যেটার নাম হবে latter এবং value নামে একটি ইন্টিজার হবে।
2. একটি কনসট্রাক্টর রাইট কর যেটা প্যারামিটার ধারণ করে এবং এটির নাম হবে latter এবং value এবং এটি ইনসট্যান্স ভ্যারিয়েবলের সূচনা করে।
3. printTile নামে একটি মেথড রাইট কর যেটা প্যারামিটার হিসাবে Tile অবজেক্ট ধারণ করে এবং পাঠক বান্ধব ফরমেটে ইনসট্যান্স ভ্যারিয়েবল প্রিন্ট করে।
4. testTile নামে একটি মেথড রাইট কর যেটা একটি Tile অবজেক্ট Z অক্ষর এবং 10 value এর সাথে তৈরি করে এবং পরে printTile ব্যবহার করে অবজেক্টের অবস্থা প্রিন্ট করে।

এই অনুশীলনীর উদ্দেশ্য হল একটি নতুন ক্লাস ডেফিনেশন তৈরি এবং কোডের মেকানিক্যাল পার্টের অনুশীলনী করা যেটা এটাকে পরীক্ষা করে।

অনুশীলনী ১১.২: Date এর জন্য ক্লাস ডেফিনেশন রাইট কর, একটি অবজেক্ট টাইপ যেটা তিনটি ইন্টিজার year, month এবং day ধারণ করে। এই ক্লাস অবশ্যই দুইটি কনসট্রাক্টর প্রদান করবে। প্রথমটি প্যারামিটার বিহীন হওয়া উচিত। দ্বিতীয়টি year, month এবং day নামে প্যারামিটার নেওয়া উচিত এবং ইনসট্যান্স ভ্যারিয়েবলের সূচনা করতে এগুলো ব্যবহার করা উচিত।

4 Scrabble is a registered trademark owned in the U.S.A and Canada by Hasbro Inc., and in the rest of the world by J.W. Spear & Sons Limited of Maidenhead, Berkshire, England, a subsidiary of Mattel Inc.

একটি main মেথড রাইট কর যা birthday নামে নতুন Date অবজেক্ট তৈরি করে। নতুন অবজেক্ট অবশ্যই তোমার birthday ধারণ করবে। তুমি যেকোন একটি কনসট্রাক্টর ব্যবহার করতে পার।

অনুশীলনী ১১.৩: একটি মূলদ সংখ্যা হল এমন একটি সংখ্যা যা দুইটি ইন্টেজারের অনুপাত বের করতে ব্যবহার করা হয়। উদাহরণস্বরূপ, 2/3 একটি মূলদ সংখ্যা, এবং তুমি 7 কে চিন্তা করতে পার একটি মূলদ সংখ্যা হিসাবে হরের অন্তর্নিহিত 1 এর সাথে। এই অ্যাসাইনমেন্টের জন্য, তোমাকে মূলদ সংখ্যার জন্য একটি ক্লাস ডেফিনেশন রাইট করতে হবে।

1. Rational.java নামে একটি নতুন প্রোগ্রাম তৈরি কর যেটা Rational নামে একটি ক্লাস ডিফাইন করে। Rational অবজেক্টের দুইটি ইন্টিজার থাকবে এবং লব ও হরকে সংরক্ষণ করতে ইনসট্যান্স ভ্যারিয়েবল সূচনা করবে।
2. একটি কনসট্রাক্টর রাইট কর যেটা কোন আর্গুমেন্ট গ্রহণ করে না এবং যেটা শূন্য থেকে দুইটি ইনসট্যান্স ভ্যারিয়েবল নির্ধারণ করে।
3. printRational নামে একটি মেথড কল কর যেটা আর্গুমেন্ট হিসাবে একটি Rational অবজেক্ট গ্রহণ করে এবং এটি কিছু যুক্তিসঙ্গত বিন্যাসে প্রিন্ট করে।
4. একটি main মেথড রাইট কর যেটা rational টাইপের সাথে একটি নতুন অবজেক্ট তৈরি করে, এর ইনসট্যান্স ভ্যারিয়েবলকে কিছু মান দ্বারা সেট কর এবং এই অবজেক্ট প্রিন্ট কর।
5. এই পর্যায়ে, তোমার একটি ক্ষুদ্র পরীক্ষা করার উপযোগী প্রোগ্রাম তৈরি হয়েছে। এটা পরীক্ষা কর এবং যদি প্রয়োজন পরে তবে ডিবাগ কর।
6. তোমার ক্লাসের জন্য একটি দ্বিতীয় কনসট্রাক্টর রাইট কর, যেটা দুইটি আর্গুমেন্ট গ্রহণ করে এবং এগুলো ইনসট্যান্স ভ্যারিয়েবলের সূচনা করতে ব্যবহার কর।
7. negate নামে একটি মেথড কল কর যেটা মূলদ সংখ্যার চিহ্নকে বিপরীত ক্রমে পরিবর্তন করে দেয়। এই মেথডের অবশ্যই একটি মডিফায়ার থাকতে হবে, যেন এটা void রিটার্ন করতে পারে। main এ লাইনগুলো যুক্ত করে নতুন মেথডটি পরীক্ষা কর।
8. invert নামে একটি মেথড কল কর যেটা সংখ্যাকে উল্টিয়ে দেয় লব ও হরের বিনিময় করে। main এ লাইনটি যুক্ত করে নতুন মেথডটি পরীক্ষা কর।
9. toDouble নামে একটি মেথড কল কর যেটা rational সংখ্যাকে double (ফ্লোটিং-পয়েন্ট সংখ্যা) সংখ্যায় রূপান্তর করে এবং ফলাফলটি রিটার্ন করে। এই মেথডটি একটি পিওর মেথড; এটি অবজেক্ট কে মডিফাই করে না। প্রতিবারের মতো এই নতুন মেথডটিও পরীক্ষা কর।
10. reduce নামে একটি মডিফায়ার রাইট কর যেটা একটি মূলদ সংখ্যাকে এর সবচেয়ে নিচের শর্তে কমিয়ে আনে এর মাধ্যমে সে খুঁজে বের করে লব ও হরের গরিষ্ঠ সাধারণ গুণনীয়ক (গসাগু) এবং বিভাজকের মাধ্যমে। এই মেথডটি অবশ্যই একটি পিওর মেথড হবে; এটা যেখানে যুক্ত হয় সেখানকার অবজেক্টের ইনসট্যান্স ভ্যারিয়েবলকে মডিফাই করে না। (গসাগু খুঁজে বের করতে অনুশীলনী ৬.১০ দেখ)।
11. add নামে একটি মেথড কল কর যেটা দুইটি rational সংখ্যাকে গ্রহণ করে আর্গুমেন্ট হিসাবে এবং নতুন আরেকটি rational অবজেক্ট রিটার্ন করে। রিটার্ন অবজেক্ট আর্গুমেন্ট এর যোগফল ধারণ করে।

   ভগ্নাংশের যোগ করার জন্য বিভিন্ন উপায় আছে। তুমি চাইলে এর যে কোন একটি ব্যবহার করতে পার। কিন্তু তোমার নিশ্চিত থাকতে হবে যেন এই কাজের রেজাল্ট যেন ক্রমেই হ্রাস পেয়ে হর ও লবের একটি কমন বিভাজক নয় হয় (1 এর চেয়ে ভিন্ন হয়)।

এই অনুশীলনীর উদ্দেশ্য হল একটি ক্লাস ডেফিনেশন রাইট করা যা মেথডের বিচিত্রতা, কন্সট্রাকটরের যুক্তকরণ, মডিফায়ার এবং পিওর ফাংশন সুংযুক্ত করে।

অধ্যায় ১২
অ্যারে

একটি অ্যারে হল ভ্যালুর একটি সেট যেখানে প্রতিটি ভ্যালু চিহ্নিত হয় একটি ইনডেক্স দ্বারা। তুমি ints, doubles, অথবা অন্য যেকোন ধরনের একটি অ্যারে তৈরি করতে পার, কিন্তু একটি অ্যারের মধ্যে সমস্ত ভ্যালু একই ধরনের হতে হবে।

শব্দবিন্যাসগত দিক থেকে, অ্যারের ধরন অন্যান্য জাভার ধরনের মতই যদি তারা [ ] দ্বারা অনুসারিত না হয়। উদাহরণস্বরূপ, int[ ] হল "পূর্ণসংখ্যার অ্যারে " এবং double[ ] হল "doubles এর অ্যারে "।

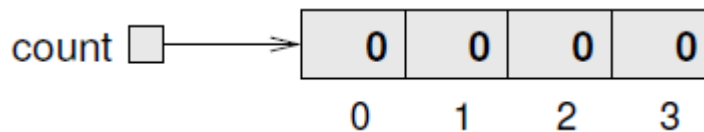
তুমি স্বাভাবিক উপায়ে এই ধরনগুলোর সাথে ভ্যারিয়েবল ডিক্লেয়ার করতে পার:

```
int[] count;
double[] values;
```

যতক্ষণ পর্যন্ত তুমি এই ভ্যারিয়েবল আরম্ভ না করবে, তারা null এ নির্ধারিত থাকবে। শুধু অ্যারে তৈরির জন্য, new ব্যবহার কর।

```
count = new int[4];
values = new double[size];
```

প্রথমে লাইনে count চারটি ইন্টিজার সম্বলিত একটি অ্যারেকে নির্দেশ করে; দ্বিতীয় লাইনে value একটি doubles সম্বলিত অ্যারেকে নির্দেশ করে। values এর উপাদানের সংখ্যা size এর উপর নির্ভর করে। তুমি যেকোন ইনটিজারকে একটি অ্যারের আকারের মত প্রকাশ করতে পার।

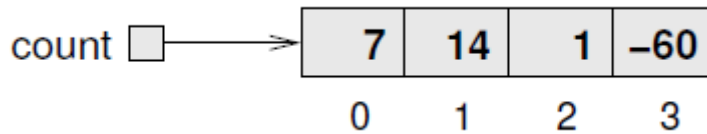
নিম্নোক্ত চিত্র প্রদর্শন করে কিভাবে স্টেট ডায়াগ্রামে অ্যারে উপস্থাপিত হয়:

বক্সের ভিতর বড় সংখ্যাগুলো হল অ্যারের উপাদান। বক্সের বাইরের ছোট সংখ্যাগুলো হল প্রতিটি বক্সের সূচক। যখন তোমার বন্টনকৃত অ্যারে যদি ints হয়, উপাদানগুলো তখন শূন্য হয়।

১২.১ উপাদানের ব্যবহার

অ্যারের মধ্যে স্টোরের মান, [ ] অপারেটর ব্যবহার করে। উদাহরণ হিসাবে count [0] নির্দেশ করে অ্যারে এর "zeroeth" উপাদানকে, এবং count [1] নির্দেশ করে "oneth" উপাদানকে। তুমি একটি এক্সপ্রেশনে [] অপারেটর যেকোন জায়গায় ব্যবহার করতে পার:

```
count[0] = 7;
```

```
count[1] = count[0] * 2;
count[2]++;
count[3] -= 60;
```

এগুলো সবই বৈধ আরোপিত কাজের স্টেটমেন্ট। এখানে কোড ফ্র্যাগমেন্টের ফলাফল দেয়া আছে:

0 থেকে 3 পর্যন্ত অ্যারে এর উপাদানগুলোর নম্বর রয়েছে, যার অর্থ হল ইনডেক্স 4 এ কোন উপাদান নেই। এটি পরিচিত লাগতে পারে, আমরা স্ট্রিং সূচকসমূহে একই জিনিস দেখেছি। অথচ, এটি একটি সাধারণ ত্রুটি অ্যারে এর সীমা অতিক্রম করার, যা একটি ArrayOutOfBoundsException নিক্ষেপ করে।

তুমি একটি সূচক হিসাবে যেকোন এক্সপ্রেশন ব্যবহার করতে পার, যতক্ষণ না পর্যন্ত এটি int টাইপ করে। একটি অ্যারে ইনডেক্স করার সবচেয়ে সাধারণ উপায় হল একটি লুপ ভ্যারিয়েবল এর সাথে করা। উদাহরণস্বরূপ:

```
int i = 0;
while (i < 4) {
      System.out.println(count[i]);
       i++;
}
```

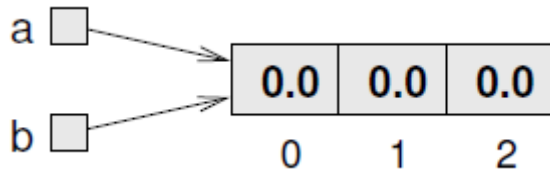
0 থেকে 4 পর্যন্ত গণনা করতে এটি হল একটি আদর্শ while লুপ, যখন লুপ ভ্যারিয়েবল i 4 হয়, তখন শর্ত বিফল হয় এবং লুপ শেষ হয়। সুতরাং লুপ সঞ্চালিত হয় তখন যখন i হয় 0, 1, 2 এবং 3 হয়।

প্রতিবার আমরা অ্যারেতে ইনডেক্স হিসাবে লুপ এর মাধ্যমে i ব্যবহার করি, i তম উপাদান প্রিন্ট করার জন্য। এই ধরনের অ্যারে ট্র্যাভেরসাল খুবই সাধারণ।

১২.২ অ্যারে অনুলিপিকরণ

যখন তুমি একটি অ্যারে ভ্যারিয়েবল অনুলিপি করতে যাবে, তুমি তখন স্মরণ রাখবে যে তুমি একটি অ্যারে এর রেফারেন্স অনুলিপি করছ। উদাহরণস্বরূপ:

```
double[] a = new double [3];
double[] b = a;
```

এই কোড তিনটি doubles এর একটি অ্যারে তৈরি করে, এবং এটি নির্দেশের জন্য দুটি ভিন্ন ভেরিয়েবল নির্ধারণ করে। এই পরিস্থিতি aliasing এর একটি ধরন।

যেকোন অ্যারে এর যেকোন পরিবর্তন অন্যটির উপর প্রভাব ফেলবে। এটি সাধারণত তুমি যা চাচ্ছ তা নয়; আরও তুমি যদি একটি নতুন অ্যারে বন্টন করতে চাও এবং একটি থেকে আরেকটিতে উপাদান অনুলিপি করতে চাও।

```
double[] b = new double [3];
int i = 0;
while (i < 4) {
    b[i] = a[i];
     i++;
 }
```

১৩.৩ অ্যারে এবং অবজেক্টসমূহ

অনেকভাবে, অ্যারে অবজেক্টের মত আচরণ করে:

- যখন তুমি অ্যারে ভ্যারিয়েবল ডিক্লেয়ার করবে, তুমি একটি অ্যারে এর একটি রেফারেন্স পাবে।
- তোমাকে new ব্যবহার করতে হবে অ্যারে নিজের থেকে তৈরি করার জন্য।
- যখন তুমি একটি আর্গুমেন্ট একটি অ্যারে হিসাবে পাস করবে, তুমি একটি রেফারেন্স পাস করবে, যার অর্থ হল যে invoked method অ্যারে এর বিষয়বস্তু পরিবর্তন করতে পারে।

আয়তক্ষেত্রের মত, কিছু অবজেক্ট আমরা যদি তাকিয়ে দেখি, তাহলে দেখব যে মান সংগ্রহের দিক থেকে তারা অ্যারে এর মতই। এখানে একটি প্রশ্ন আসে, “কিভাবে একটি 4 ইনটিজারের একটি অ্যারে একটি আয়তক্ষেত্র থেকে আলাদা?”

তুমি যদি অধ্যায়ের শুরুতে “অ্যারে ” এর সংজ্ঞায় ফিরে যাও, তুমি একটি পার্থক্য দেখবে: একটি অ্যারে এর উপাদানসমূহ সূচক দ্বারা নির্দেশিত হয়, এবং একটি অবজেক্টের উপাদানসমূহের নাম আছে।

অন্য পার্থক্য হল যে একটি অ্যারে এর উপাদানসমূহ একই ধরণের হতে হবে। অবজেক্টের বিভিন্ন ধরণের instance ভ্যারিয়েবল থাকতে পারে।

১২.৪ লুপ এর জন্য

আমরা সাধারণত যে লুপসমূহ লিখি তাদের কিছু উপাদান একই থাকে। তাদের সবগুলোই একটি ভ্যারিয়েবল দ্বারা শুরু হয়; তাদের একটি পরীক্ষা অথবা শর্ত থাকে; যা নির্ভর করে ঐ ভ্যারিয়েবল এর উপরে; এবং লুপ এর ভেতরে ঐ ভ্যারিয়েবলে তারা যা করে, যদি তারা উন্নয়ন করতে চায়।

এই ধরনের লুপ খুবই সাধারণ তাই অন্য একটি লুপ স্টেটমেন্ট রয়েছে, যাকে for বলা হয়, তা আরো সংক্ষিপ্তাকারে প্রকাশ করে। সাধারণ সিনট্যাক্স এই ধরনের দেখতে হয়:

```
for (INITIALIZER; CONDITION; INCREMENTOR) {
    BODY
}
```

এই স্টেটমেন্টটির সমকক্ষ হল

```
INITIALIZER;
while (CONDITION) {
        BODY
        INCREMENTOR
}
```

এটি আরো সংক্ষিপ্ত ব্যতিত এবং, এটি সমস্ত লুপ-সংক্রান্ত স্টেটমেন্ট এক জায়গায় একত্রকরণ করার জন্য, এটি পড়তে সহজ হয়। উদাহরণ স্বরুপ:

```
for (int i = 0; i < 4; i++) {
    System.out.println(count[i]);
}
```

এর সমকক্ষ হল

```
int i = 0;
while (i < 4) {
     System.out.println(count[i]);
      i++;
}
```

১২.৫ অ্যারের দৈর্ঘ্য

সমস্ত অ্যারের একটি একই নামের ইন্সট্যান্স ভ্যারিয়েবল আছে: এটি হল length। এটি আশ্চর্যজনক নয়, এটি অ্যারের দৈর্ঘ্য ধারণ করে (উপাদানসমূহের সংখ্যা)। একটি কনস্ট্যান্ট ভ্যালুর থেকে, একটি লুপ এর ঊর্ধ্বসীমা মানের এই মান ব্যবহার এটি একটি ভাল ধারণা। ঐভাবে, যদি অ্যারের আকার পরিবর্তিত হয়, তোমাকে সমস্ত লুপ পরিবর্তন করার প্রোগ্রামের মধ্য দিয়ে যেতে হবে না; তারা যেকোন আকারের অ্যারেতে সঠিকভাবে কাজ করবে।

```
for (int i = 0; i < a.length; i++) {
    b[i] = a[i];
}
```

শেষবারে যখন লুপ এর বডি সঞ্চালিত হল, i হল a.length – 1, যা হল শেষ উপাদানের ইনডেক্স। যখন i হল a.length এর সমান, শর্ত ব্যর্থ হয় এবং বডি সঞ্চালিত না হয়, যা একটি চমৎকার জিনিস, যতক্ষণ পর্যন্ত না এটি

একটি ব্যতিক্রম করে। এই কোড থেকে ধারণা করা যায় যে অ্যারে b কমপক্ষে a এর সমান উপাদান ধারণ করে।

১২.৬ র‍্যানডম সংখ্যা

অনেক কম্পিউটার প্রোগ্রাম যতবার তাদের নির্বাহ করা হয় ততবার একই জিনিস করে, তাই তাদের নির্ণায়ক বলা যেতে পারে। সাধারণত, নির্ণায়ক একটি চমৎকার জিনিস, যদি আমরা একই ফলাফল একই গণনার জন্য প্রত্যাশা করি। কিন্তু কিছু অ্যাপ্লিকেশনের জন্য আমরা চাই কম্পিউটার অনির্দেশ্য থাকুক। গেম একটি সুস্পষ্ট উদাহরণ, কিন্তু সেখানে আরো আছে।

এটি প্রোগ্রাম তৈরি করে অনির্ধারিত সক্রিয় ফলাফল পাওয়া সত্যিই সহজ নয়, কিন্তু সেখানে কিছু উপায় আছে যার জন্য এটিকে অন্ততপক্ষে অনির্ধারিত মনে হয়। তাদের মধ্যে একটি হল র‍্যানডম নাম্বার তৈরি করা এবং তাদেরকে প্রোগ্রামের ফলাফল নির্ধারণের জন্য ব্যবহার করা। জাভা একটি মেথড প্রদান করে যা pseudorandom নম্বর তৈরি করে, যা সত্যিই র‍্যানডম নাও হতে পারে, কিন্তু আমাদের উদ্দেশ্যে, তারা করে।

অংক ক্লাশে র‍্যানডম মেথড এর নথি দেখ। 0.0 এবং 1.0 এর মধ্যে রিটার্ন ভ্যালু হল double। সুনির্দিষ্ট করে বলতে গেলে এটি হল 0.0 এর বৃহত্তর অথবা সমান এবং অবশ্যই 1.0 এর ক্ষুদ্রতর। তুমি যতবার র‍্যানডম করবে তুমি pseudorandom সিকুয়েন্স এর পরবর্তী নম্বর পাবে। একটি নমুনা দেখার জন্য, এই লুপ রান কর:

```
for (int i = 0; i < 10; i++) {
        double x = Math.random();
        System.out.println(x);
}
```

এবং high এর মত আপার বাউন্ডের মধ্যে র‍্যানডম ডাবল তৈরির জন্য, তুমি x কে high দ্বারা গুন দিতে পার।

১২.৭ র‍্যানডম সংখ্যাসমূহের অ্যারে

কিভাবে তুমি low এবং high এর মধ্যে একটি র‍্যানডম ইনটিজার তৈরি করবে? যদি তোমার randomInt এর প্রয়োগ সঠিক হয়, তখন low থেকে high-1 এর মধ্যে প্রতিটি মানের সমান সম্ভাব্যতা থাকা উচিৎ। যদি তুমি একটি দীর্ঘ নম্বরের সিরিজ তৈরি কর, তাহলে প্রতিটি মান প্রদর্শিত হওয়া উচিৎ, কমপক্ষে আনুমানিক, বারের সমান সংখ্যার।

তোমার মেথড পরীক্ষা করার একটি উপায় হল র‍্যানডম ভ্যালুর একটি বিশাল সংখ্যা তৈরি কর, একটি অ্যারেতে সংরক্ষণ কর, এবং প্রতিটি ভ্যালু ঘটার সময় গণনা কর।

নিম্নলিখিত পদ্ধতি একটি একক আর্গুমেন্ট, অ্যারের আকারে নেয়। এটি পূর্ণসংখ্যার একটি নতুন অ্যারের স্থান নির্দেশ করে, এটিকে র‍্যানডম ভ্যালুর সাথে পূর্ণ করে, এবং নতুন অ্যারের একটি রেফারেন্স উৎপন্ন করে।

```
public static int[] randomArray(int n) {
    int[] a = new int[n];
    for (int i = 0; i<a.length; i++) {
        a[i] = randomInt(0, 100);
    }
```

```
        return a;
    }
```

রিটার্ণ টাইপ হল int [], যার অর্থ হল যে এই মেথড পূর্ণসংখ্যার একটি অ্যারে উৎপন্ন করে। এই মেথডটি পরীক্ষা করতে, সুবিধাজনক হল এমন একটি মেথড যা একটি অ্যারের বিষয়বস্তু মুদ্রণ করে।

```
    public static void printArray(int[] a) {
        for (int i = 0; i<a.length; i++) {
            System.out.println(a[i]);
        }
    }
```

নিম্নলিখিত কোড একটি অ্যারে তৈরি করে এবং এটি মুদ্রণ কর:

```
        int numValues = 8;
        int[] array = randomArray(numValues);
        printArray(array);
```

আমার একটি মেশিনের আউটপুট হল

```
27
6
54
62
54
2
44
81
```

এটি সুন্দর র‍্যানডম দেখতে। আপনার ফলাফল ভিন্ন হতে পারে।

যদি এটি পরীক্ষার সোর্স হয় (এবং সেগুলো হতে পারে বেশ খারাপ পরীক্ষার সোর্স), শিক্ষক হয়তবা একটি histogram ফর্মে ক্লাশে ফলাফল উপস্থাপন করতে পারেন, এটি হল কাউন্টারের একটি সেট যা প্রতিটি ভ্যালু ঘটার সময়ের নম্বরের ট্র্যাক রাখে।

পরীক্ষার স্কোরের জন্য, আমাদের দশটি কাউন্টারের প্রয়োজন কতজন ছাত্র 90s, 80s, ইত্যাদি নম্বর প্রাপ্ত হয়েছে তা ট্র্যাক করতে। পরবর্তী কিছু সেকশনে একটি হিস্টোগ্রাম তৈরির জন্য কোড ডেভেলপ করা হবে।

১২.৮ গণনা

এটির মত সমস্যার উপায় হল সাধারণ মেথড চিন্তা করা যা লিখতে সহজ, এরপর একটি সমাধানে একত্রিত করা। এই প্রক্রিয়াকে বলা হয় bottom-up development. দেখুন
http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design.

এটি সবসময় সুস্পষ্ট নয় যে কোথা থেকে শুরু করতে হবে, কিন্তু একটি ভাল পদ্ধতি হল সাব প্রবলেম খোঁজা যা তুমি পূর্বে দেখেছ এরকম একটি প্যাটার্ন ফিট করা।

সেকশন ৮.৭ আমারা একটি লুপ দেখেছি যা একটি স্ট্রিংকে বাধা দেয় এবং প্রদত্ত একটি অক্ষরের প্রদর্শনের সময়ের সংখ্যা গণনা করে। এই প্রোগ্রামটিকে তুমি "traverse and count" নামক একটি প্যাটার্ণের উদাহরণ হিসাবে চিন্তা করতে পার। এই প্যাটার্ণের উপাদানগুলো হল:

- একটি সেট অথবা কন্টেইনার, যা অ্যারে অথবা স্ট্রিং এর মত আড়াআড়িভাবে উপস্থাপিত হতে পারে।
- একটি পরীক্ষা যা তুমি কন্টেইনারের প্রতিটি উপাদানে প্রয়োগ করতে পার।
- একটি কাউন্টার যা কতটি উপাদান টেস্ট পাস করে তার ট্রাক পাস করে।

এই অবস্থায়, কন্টেইনার হল একটি পূর্ণসংখ্যার একটি অ্যারে। টেস্টটি একটি প্রদত্ত ভ্যালু সীমার মধ্যে পড়তে পারে আবার নাও পারে।

এখানে একটি inRange মেথড আছে যা একটি অ্যারের উপাদানের নম্বর গণনা করে এবং প্রদত্ত সীমার মধ্যে পড়ে। প্যারামিটারগুলো হল অ্যারে এবং দুটি ইন্টিজার যা সীমার নিম্ন এবং উচ্চ সীমা নির্দিষ্ট করে।

```
public static int inRange(int[] a, int low, int high) {
    int count = 0;
    for (int i = 0; i < a.length; i++) {
        if (a[i] >= low && a[i] < high) count++;
    }
    return count;
}
```

আমি আসলে নিশ্চিত না সীমার মধ্যে সত্যিই কিছু নিম্ন অথবা উচ্চ এর সমান কিনা, কিন্তু তুমি কোড থেকে দেখতে পার যে low হল in এবং high হল out. যা আমাদের যেকোন উপাদান দুবার গণনা করা থেকে বিরত রাখে।

এখন আমরা সীমার মধ্যে আমরা যেটিতে আগ্রহী তার স্কোরের নম্বর গণনা করব:

```
int[] scores = randomArray(30);
int a = inRange(scores, 90, 100);
int b = inRange(scores, 80, 90);
int c = inRange(scores, 70, 80);
int d = inRange(scores, 60, 70);
int f = inRange(scores, 0, 60);
```

১২.৯ হিস্টোগ্রাম

এই কোড হল পুনরাবৃত্তিকারী, কিন্তু এটি গ্রহণযোগ্য যতক্ষণ নম্বরের সীমা কম থাকে। কিন্তু ধারণা কর যে আমরা প্রতিটি স্কোর প্রদর্শনের সময়ের নম্বরের ট্রাক রাখতে চাই, সমস্ত 100 সম্ভাব্য ভ্যালুর। তুমি কি এটি লিখতে চাও?

```
int count0 = inRange(scores, 0, 1);
int count1 = inRange(scores, 1, 2);
int count2 = inRange(scores, 2, 3);
...
int count3 = inRange(scores, 99, 100);
```

আমি তা মনে করি না। আমরা আসলে একটি উপায় চাই 100 পূর্ণসংখ্যা সংরক্ষণ করার, বিশেষ করে তাই আমরা প্রতিটি চালানোর জন্য একটি ইনডেক্স ব্যবহার করতে পারি। Hint: array.

গণনার প্যাটার্ণ হল একই আমরা একটি একক কাউন্টার অথবা কাউন্টারের একটি অ্যারে ব্যবহার করতে পারি। এই ক্ষেত্রে, আমরা অ্যারের বাইরে লুপ আরম্ভ করতে পারি; এরপর, লুপ এর ভিতরে, আমরা আহ্বান করব এবং ফলাফল সংরক্ষণ করব:

```
int[] counts = new int[100];
for (int i = 0; i < counts.length; i++) {
    counts[i] = inRange(scores, i, i+1);
}
```

এখানে শুধুমাত্র কৌশলের বিষয় হল যে আমরা দুটি রোলে লুপ ভ্যারিয়েবল ব্যবহার করছি: অ্যারের মধ্যে ইনডেক্স হিসাবে, এবং inRange এ প্যারামিটার হিসাবে।

১২.১০ একটি single-pass সমাধান

এই কোড কাজ করছে, কিন্তু এটি ততটা কার্যকর নয় যতটা হওয়া দরকার ছিল। যতবার এটি inRange আহবান করে, এটি সম্পূর্ণ অ্যারে অতিক্রম করে। নম্বর এর সীমা বৃদ্ধি পাচ্ছে, অনেক ট্র্যাভারসাল হয়ে যা হয়।

এটি হয়ত ভাল একটি অ্যারের মাধ্যমে একক পাস তৈরি করা, এবং প্রতিটি ভ্যালুর জন্য যে সীমার মধ্যে পড়ে তা গণনা করা। তারপর আমরা উপযুক্ত কাউন্টার বাড়াতে পারি। এই উদাহরণে, ঐ গণনা হল নগন্য, কারণ আমরা কাউন্টারের অ্যারেতে ইনডেক্স হিসাবে ভ্যালুটিকে ব্যবহার করতে পারি।

এখানে কোডটি আছে যা স্কোরের একটি অ্যারে একবার অতিক্রম করে এবং একটি হিস্টোগ্রাম তৈরি করে।

```
int[] counts = new int[100];
for (int i = 0; i < scores.length; i++) {
    int index = scores[i];
    counts[index]++;
}
```

১২.১১ শব্দকোষ

অ্যারে: এটি হল ভ্যালুর একটি সম্ভার, যেখানে সমস্ত ভ্যালু একই ধরনের হয়, এবং প্রতিটি ভ্যালু একটি ইনডেক্স দ্বারা নির্দিষ্ট হয়।

ইলিমেন্ট: একটি অ্যারের মধ্যে একটি ভ্যালু। [] অপারেটর ইলিমেন্ট নির্বাচন করে।

ইনডেক্স: একটি পূর্ণসংখ্যা ভ্যারিয়েবল অথবা ভ্যালু যা একটি অ্যারের উপাদান নির্দেশ করে।

নির্ণায়ক: একটি প্রোগ্রাম যা প্রতিবার একই জিনিস করে যখন একে আহবান করা হয়।

সুডোর‍্যানডম: নম্বরের একটি অনুক্রম যা র‍্যানডম ভাবে প্রদর্শিত হবে, কিন্তু আসলে এটি নির্ণয়ক গণনার একটি ফলাফল।

হিস্টোগ্রাম: পূর্ণসংখ্যার একটি অ্যারে যেখানে প্রতিটি ইনটিজার ভ্যালুর সংখ্যা গণনা করে যা একটি নির্দিষ্ট সীমার মধ্যে পড়ে।

১২.১২ অনুশীলনী

অনুশীলনী ১২.১: CloneArray নামক একটি মেথড লেখ যার মধ্যে একটি প্যারামিটার হিসাবে একটি অ্যারে থাকবে, একই আকারের একটি নতুন অ্যারে তৈরি করা হবে, প্রথম অ্যারের ইলিমেন্ট নতুনটিতে অনুলিপি করা হবে, এবং তারপর নতুন অ্যারেতে একটি রেফারেন্স ফেরত আনা হবে।

অনুশীলনী ১২.২: RandomDouble নামক একটি মেথড লেখ যা দুটি doubles নিবে, নিম্ন এবং উচ্চ, এবং সেটি একটি random double x ফেরত দিবে যেহেতু low $\leq$ x < high.

অনুশীলনী ১২.৩: RandomInt নামক একটি মেথড লেখ যা দুটি আর্গুমেন্ট নেয়, উচ্চ এবং নিম্ন, এবং যা উচ্চ এবং নিম্ন এর মধ্যে একটি র‍্যানডম ইন্টিজার ফেরত দেয়, উচ্চ ব্যতিত।

অনুশীলনী ১২.৪: সেকশন ১২.১০ এর কোড makeHist নামক একটি মেথডে এনক্যাপসুলেট কর যা স্কোরের একটি অ্যারে নিবে এবং একটি অ্যারে এর ভ্যালুর একটি হিস্টোগ্রাম ফেরত দিবে।

অনুশীলনী ১২.৫: areFactors নামক একটি মেথড লেখ যা একটি ইন্টিজার n এবং ইন্টিজারসমূহের একটি অ্যারে নেয়, এবং যা true রিটার্ণ করবে যদি অ্যারে এর নম্বরগুলো n এর ফ্যাক্টর হয় (যাকে বলা যায় n তাদের সকল দ্বারা বিভাজ্য)। ইঙ্গিত: অনুশীলনী ৬.১. দেখুন।

অনুশীলনী ১২.৬: একটি মেথড লিখ যা ইনটিজারের একটি অ্যারে নেয় এবং আর্গুমেন্টকে টার্গেট করে নামকরণকৃত একটি ইনটিজার নাও, এবং সেটি প্রথম ইনডেক্স রিটার্ন করে যেখানে অ্যারেতে টার্গেট প্রদর্শিত হয়, যদি এটি তা করে, এবং না করলে -1 হবে।

অনুশীলনী ১২.৭: ভ্যারিয়েবল এবং মেথড নাম অর্থপূর্ণ থাকা উচিৎ সেজন্য কিছু প্রোগ্রামার সাধারণ নিয়মে অসম্মত হয়। পরিবর্তে তারা চিন্তা করে ফ্রুট এর পরে ভ্যারিয়েবল এবং মেথডের নামকরণ করা উচিৎ।
নিম্নের প্রতিটি মেথডের জন্য, একটি বাক্য লেখ যার মধ্যে মেথডটি কি করে তার বিবরণ থাকবে। প্রতিটি ভ্যারিয়েবল এর জন্য, সেটির কাজের ভূমিকা নির্দিষ্ট কর।

```
public static int banana(int[] a) {
     int grape = 0;
     int i = 0;
    while (i < a.length) {
         grape = grape + a[i];
```

```
        i++;
    }
    return grape;
}

public static int apple(int[] a, int p) {
    int i = 0;
    int pear = 0;
    while (i < a.length) {
        if (a[i] == p) pear++;
        i++;
    }
    return pear;
}

public static int grapefruit(int[] a, int p) {
    for (int i = 0; i<a.length; i++) {
        if (a[i] == p) return i;
    }
    return -1;
}
```

এই অনুশীলনীর উদ্দেশ্য হল কোড পড়া অনুশীলন করা এবং আমরা যেসকল গণনার প্যাটার্ণ দেখেছি তা চিনতে পারা।

অনুশীলনী ১২.৮:

১. নিম্নের প্রোগ্রামের ফলাফল কি?

২. একটি স্ট্যাক ডায়াগ্রাম তৈরি কর যা mus রিটার্ণের পূর্বেই প্রোগ্রামের স্টেট প্রদর্শন করবে।

৩. mus যা করে অল্প কথায় তার একটি বিবরণ দাও।

```
public static int[] make(int n) {
    int[] a = new int[n];
    for (int i = 0; i < n; i++) {
        a[i] = i+1;
    }
    return a;
}

public static void dub(int[] jub) {
    for (int i = 0; i < jub.length; i++) {
```

```
            jub[i] *= 2;
        }
}

public static int mus(int[] zoo) {
        int fus = 0;
        for (int i = 0; i < zoo.length; i++) {
            fus = fus + zoo[i];
        }
        return fus;
}

public static void main(String[] args) {
        int[] bob = make(5);
        dub(bob);
        System.out.println(mus(bob));
}
```

অনুশীলনী ১২.৯: আমরা ট্রাভারসিং অ্যারের জন্য এমন অনেক প্যাটার্ণ দেখেছি যা রিকারসিভ ভাবেও লেখা যায়। এটি সেরকম প্রচলিত নয়, কিন্তু এটি একটি প্রয়োজনীয় অনুশীলন।

১. maxInrange নামক একটি মেথড লেখ যা ইন্টিজারের একটি অ্যারে এবং সূচকসমূহের একটি রেঞ্জ নেবে (lowIndex এবং highIndex), এবং যা অ্যারেতে সর্বোচ্চ ভ্যালু খুঁজে বের করবে, শুধুমাত্র lowIndex এবং highIndex এর উপাদান উভয় শেষ সহ বিবেচনা করে।

এই মেথডটি রিকারসিভ হওয়া উচিৎ। যদি রেঞ্জ এর দৈর্ঘ্য 1 হয়, সেটি হল, lowIndex == highIndex হলে, আমরা তাড়াতাড়ি জানব যে রেঞ্জের মধ্যে একমাত্র উপাদান সর্বাধিক হবে। তাই এটি হল বেস কেস।

যদি সেখানে রেঞ্জে একের অধিক উপাদান থাকে, আমরা অ্যারেটি দুটি ভাগে বিভক্ত করতে পারি, প্রতিটি থেকে সর্বোচ্চ পেতে পারি,এবং তারপর আমরা maxima এর সর্বোচ্চ পেতে পারি।

২. maxInRange এর মত মেথড ব্যবহার বেমানান হতে পারে। একটি অ্যারেতে সর্বোচ্চ উপাদান বের করতে আমাদের একটি রেঞ্জ প্রদান করতে হবে যা সম্পূর্ণ অ্যারে এর অন্তর্ভুক্ত।

```
double max = maxInRange(array, 0, a.length-1);
```

max নামক একটি মেথড লেখ যা একটি অ্যারেকে একটি প্যারামিটার হিসাবে নেয় এবং যা সন্ধান এবং সর্বোচ্চ ভ্যালু রিটার্নের জন্য maxInRange ব্যবহার করে। max এর মত মেথডকে কিছু কিছু সময়ে wrapper methods বলা হয় কারণ তারা একটি অসুবিধাজনক স্তরের কাছাকাছি বিমূর্তন স্তর প্রদান করে এবং এটির ব্যবহার সুবিধাজনক করে। যে মেথডটি আসলে সম্পূর্ণ গণনাটি সম্পাদন করে তাকে helper method বলা হয়।

৩. wrapper-helper প্যাটার্ণ ব্যবহার করে find এর রিকারসিভ সংস্করণ লেখ। find ইন্টিজারের একটি অ্যারে নিতে অথবা একটি টার্গেট ইন্টিজার নিতে পারে। এটি হয়ত প্রথম লোকেশনের ইনডেক্স রিটার্ণ করতে পারে যেখানে

টার্গেট ইন্টিজার অ্যারেতে প্রদর্শিত হয়, অথবা যদি তা প্রদর্শিত না হয় তাহলে -1 হয়।

অনুশীলনী ১২.১০: সবচেয়ে বড় ইলিমেন্ট খোঁজার জন্য একটি অ্যারে এর ইলিমেন্ট বাছাইয়ের জন্য One খুব কার্যকারী উপায় নয় এবং এটি প্রথম ইলিমেন্টে সোয়াপ করা হবে, এরপর দ্বিতীয়-বৃহত্তম ইলিমেন্ট খোঁজা হবে এবং দ্বিতীয়টিতে সোয়াপ করা হবে, এবং এমনিভাবেই চলতে থাকবে। এই মেথডটিকে একটি selection sort বলা হয় (http://en.wikipedia.org/wiki/Selection_sort দেখুন)।

১. indexOfMaxInRange নামক একটি মেথড লেখ যা ইনটিজারের একটি অ্যারে নিবে, প্রদত্ত রেঞ্জের মধ্যে বৃহত্তম ইলিমেন্ট খুঁজে বের করবে, এবং এটির index রিটার্ন করবে। তুমি তোমার maxInRange এর রিকারসিভ সংস্করণ পরিবর্তন করতে পারবে অথবা তুমি স্ক্র্যাচ এর একটি মিথস্ক্রিয়সংস্করণ লিখতে পার।

২. swapElement নামক একটি মেথড লেখ যা ইন্টিজারের একটি অ্যারে এবং দুটি সূচক নিবে, এবং প্রদত্ত সূচকসমূহে ইলিমেন্টগুলো সোয়াপ করবে।

৩. selectionSort নামক একটি মেথড লেখ যা ইন্টিজারের একটি অ্যারে নিবে এবং indexOfMaxInRange ও swapElement ব্যবহার করে অ্যারে বড় থেকে ছোট সাজাতে সাহায্য করবে।

অনুশীলনী ১২.১১: LetterHist নামক একটি মেথড লেখ একটি স্ট্রিংকে একটি প্যারামিটার হিসাবে নেয় এবং যা স্ট্রিং এ লেটারের একটি হিস্টোগ্রাম রিটার্ন করে। হিস্টোগ্রামের শূন্যতম উপাদান স্ট্রিং এ a এর নম্বর ধারণ করে (উচ্চ এবং নিম্ন ক্ষেত্রের); ২৫তম ইলিমেন্ট z এর নম্বর ধারণ করা উচিৎ। তোমার সমাধান শুধুমাত্র একবার স্ট্রিংটি অতিক্রম করা উচিৎ।

অনুশীলনী ১২.১২: একটি শব্দকে "doubloon" বলা যায় যদি প্রতিটি অক্ষর সঠিকভাবে দুবার প্রদর্শিত হয়। উদাহরণস্বরূপ, আমি আমার ডিকশনারিতে নিম্নের doubloons গুলো খুঁজে পেয়েছি।

> Abba, Anna, appall, appearer, appeases, arraigning, beriberi,
> bilabial, boob, Caucasus, coco, Dada, deed, Emmett, Hannah,
> horseshoer, intestines, Isis, mama, Mimi, murmur, noon, Otto,
> papa, peep, reappear, redder, sees, Shanghaiings, Toto

isDoubloon নামক একটি মেথড লেখ যা সঠিক রির্টার্ন দেয় যদি প্রদত্ত শব্দটি doubloon হয় অন্যথায় মিথ্যা হয়।

অনুশীলনী ১২.১৩: দুটি শব্দ হল anagrams যদি তারা একই অক্ষর ধারণ করে (এবং প্রতিটি অক্ষরের একই নম্বর)।

উদাহরণ হিসাবে, "stop" হল "pots" এর একটি anagram এবং "allen downey" হল "well annoyed" এর একটি anagram।

একটি মেথড লেখ যা দুটি স্ট্রিং নেয় এবং true রিটার্ণ করে যদি স্ট্রিং দুটি পরস্পর এনাগ্রাম হয়।

ঐচ্ছিক চ্যালেঞ্জ: শুধুমাত্র একবার স্ট্রিং এর অক্ষরগুলো পড়া হবে।

অনুশীলনী ১২.১৪. Scrabble এ প্রতিটি প্লেয়ারের তদের অক্ষরের সম্বলিত এক সেট টাইলস আছে এবং

গেমটির উদ্দেশ্য হল ঐ অক্ষরগুলো ব্যবহার করে শব্দ ব্যাখ্যা করা। স্কোরিং সিস্টেম জটিল, কিন্তু বড় শব্দ সাধারণত ছোট শব্দের থেকে গুরুত্বপূর্ণ।

কল্পনা কর "quijibo" এর মত একটি স্ট্রিং তোমাকে তোমার এক সেট টাইলস দেয়া হয়েছে এবং তোমাকে "jib" এর মত অন্য একটি স্ট্রিং পরীক্ষার জন্য দেয়া হয়েছে। canSpell নামক একটি মেথড লেখ যা দুটি স্ট্রিং নেয় এবং true রিটার্ন দেয় যদি টাইলস এর সেট শব্দটি ব্যাখ্যা করতে ব্যবহৃত হতে পারে। তোমার হয়ত একই অক্ষরের একের অধিক টাইল থাকতে পারে, কিন্তু তুমি শুধুমাত্র একবার টাইল ব্যবহার করতে পারবে।

ঐচ্ছিক চ্যালেঞ্জ: শুধুমাত্র একবার স্ট্রিং এর অক্ষরগুলো পড়া হবে।

অনুশীলনী ১২.১৫: Scrabble এ প্রকৃতপক্ষে, কিছু খালি টাইল আছে যা ওয়াইল্ড কার্ড হিসাবে ব্যবহৃত হতে পারে; যা হল; একটি খালি টাইল যেটি যেকোন অক্ষরের পরিবর্তে ব্যবহৃত হতে পারে।

canSpell এর জন্য একটি অ্যালগরিদম চিন্তা কর যা ওয়াইল্ড কার্ডের সাথে ডিল করে। কিভাবে ওয়াইল্ড কার্ড উপস্থাপন করা হয়েছে এ ধরনের প্রয়োগের বিস্তারিত দেয়ার দরকার নেই। শুধুমাত্র অ্যালগরিদম, ইংরেজি এর ব্যবহার, pseudocode, অথবা জাভা বর্ণনা কর।

অধ্যায় ১৩

অবজেক্ট এর অ্যারে (Arrays of Objects)

১৩.১ সামনে রয়েছে পথ (The Road Ahead)

পরের তিনটি অধ্যায়ে আমরা এমন প্রোগ্রাম তৈরির চেষ্টা করবো যা কার্ড খেলার (playing cards) এবং decks of cards এর সাথে কাজ করবে। এই কাজ করার পূর্বেই আমরা দেখে নেই আমাদের বিভিন্ন ধাপে কি কি কাজ করতে হবে।

- এই অধ্যায়ে আমরা একটি Card class এবং write methods নির্দেশ করবো যা Cards এবং Cards এর arrays (arrays of Cards) নিয়ে কাজ করবে।
- অধ্যায় ১৪ তে আমরা Deck class এবং write methods তৈরি করবো যা Decks এর ওপর অপারেট করবে।
- অধ্যায় ১৫ তে আমরা object-oriented programming (OOP) দেখবো এবং আমার আমাদের Card and Deck classes গুলোকে OOP style-এ রূপান্তরের চেষ্টা করবো।

এইভাবে কাজ করতে পারলে আমার আমাদের পথকে সহজতর করতে পারবো। তবে এক্ষেত্রে drawback বা সমস্যা হলো এখানে আমরা একই কোডের বিভিন্ন ভার্সন দেখতে পাবো। যা অনেক সময়ই আমাদের জন্য বিভ্রান্তিকর হতে পারে। যদি এটি সাহায্য করে, তাহলে আমরা এই বই-এ উল্লেখিত সব অধ্যায়ের কোডগুলোই ডাউনলোড করতে পারি নিচের লিংক থেকে। http://thinkapjava.com/code/Card1.java.

১৩.২ কার্ড অবজেক্ট (Card objects)

যদি তোমরা সাধারণ কার্ড খেলার সাথে পরিচিত না হও তাহলে এটিই আমাদের জন্য ভালো সময় এখানে একটি deck নেওয়া, অন্যথায় এই অধ্যায় আলাদা কোন অর্থ বহন করবে না। অথবা আমরা পড়তেও পারি এই লিংক থেকে http://en.wikipedia.org/wiki/Playing_card ।

একটি ডেকে 52 টি কার্ড রয়েছে; প্রত্যকেটি কোন না কোন চার suits এর অওত্তাভুক্ত এবং13 ranks এর অন্তর্ভুক্ত। Suits গুলো হলো Spades/স্প্যাড, Hearts/হার্ট, Diamonds/ডায়মন্ড এবং Clubs/ক্লাব (Bridge এর অধোগামী/descending অনুসারে)। আর rank গুলো হলো Ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen এবং King। কিধরনের খেলা হচ্ছে তার ওপর নর্ভর করে Ace কখনো King এর চেয়ে বড় হতে পারে আবার কখনো 2 এর চেয়ে ছোটও হতে পারে।

যদি আমরা খেলার কার্ডটি নতুন একটি অবজেক্ট দিয়ে নির্দেশ করে প্রতিস্থাপন করতে চাই, তাহলে ভালো হয় rank এবং suit এর কোনটি instance variables তা উল্লেখ করা। Instance variables নির্দেশ করা আবশ্যিক কোন কাজ নয়। এক্ষেত্রে একটি সম্ভবনা থাকে Strings এ, যা suits এর জন্য "Spade" এর মতো বিষয় ধারণ করে এবং ranks এর জন্য "Queen" ধারণ করে। এর বাস্তবায়নের একটি সমস্যা হলো rank অথবা suit এর কোন কার্ডটি বড় তা তুলনা করা সহজ নয়।

Ranks এবং suits কে encode হিসেবে ব্যবহার করা আমাদের জন্য বিকল্প উপায় হতে পারে। "encode" এর মাধ্যমে আমরা আসলে সবাই যা বোঝে তা বোঝাতে চাচ্ছি না। "encode" বলতে সাধারণত সবাই বোঝে কোন গোপন কোডে encrypt করা বা ভাষান্তর করা। আর কম্পিউটার বিজ্ঞানের ভাষায় আমরা বুঝি "encode" হলো এমন কিছু যা "নম্বরের ক্রম বা sequence of numbers এবং নির্দিষ্ট বস্তর মধ্যে একটি সুনির্দিষ্ট নির্দেশনা বা mapping বের করে ঠিক যেভাবে আমরা উপস্থাপন করতে চাই সেভাবে।" উদাহরণস্বরূপ,

Spades	$\mapsto$	3
Hearts	$\mapsto$	2
Diamonds	$\mapsto$	1
Clubs	$\mapsto$	0

এই mapping এর আবশ্যকিই বৈশিষ্ট্য হচ্ছে তা ক্রমানুসারে suits map থেকে integers এর দিকে যাবে। যার ফলে আমরা integers এর তুলনার মাধ্যমে আমরা suits এর তুলনা করতে পারবো। ranks এর mapping করা আবশ্যিক; প্রত্যেকটি numerical বা গাণিতিক ranks maps তার অনুরূপ integer এর সমরূপ এবং একই রকম face cards এর জন্য:

Jack	$\mapsto$	11
Queen	$\mapsto$	12
King	$\mapsto$	13

যে কারণে আমি এই mappings এ mathematical notation ব্যবহার করছি তা হলো এগুলো এই প্রোগ্রামের অংশ নয়। তারা মূলত প্রোগ্রাম ডিজাইনের অংশ কিন্তু তারা কখনো স্পষ্টভাবে কোডে দৃশ্যমান থাকে না। Card type এর class এর সংজ্ঞা আমরা হবে নিম্নরূপ:

```
class Card
{
        int suit, rank;
        public Card() {
                this.suit = 0; this.rank = 0;
        }

        public Card(int suit, int rank) {
                this.suit = suit; this.rank = rank;
        }
}
```

স্বাভাবিক নিয়মে, আমি দুটি constructors নিয়েছি: একটি প্রত্যেক instance variable এর প্যারামিটার হিসেবে নেওয়া হয়েছে; এবং অন্যটি কোন প্যারামিটার গ্রহণ করে নি।

তিনটি Clubs কে উপস্থাপন করার জন্য আমাদের একটি অবজেক্ট তৈরি করতে হবে, এর জন্য আমাদের invoke করতে হবে new:

```
Card threeOfClubs = new Card(0, 3);
```

প্রথম argument, 0 প্রতিস্থাপন করবে suit Clubs কে।

১৩.৩ প্রিন্ট কার্ড মেথড (The printCard method)

যখন আমরা একটি new class তৈরি করবো, প্রথম ধাপ হলো instance variables, declare করা এবং constructors লেখা। দ্বিতীয় ধাপ হলো standard methods লেখা, যা প্রতিটি অবজেক্টের থাকতে হয়, এর মধ্যে অন্তর্ভুক্ত থাকবে অবজেক্টকে প্রিন্ট করার methods এবং এমন দুটি বা তিনটি methods যা অবজেক্টগুলোকে তুলনা করবে। চলো আমরা printCard দিয়ে শুরু করি।

print Card objects হলো এমন একটি উপায় যাতে মানুষ সহজেই তা পড়তে পারে। আমরা শব্দের সম্মুখে integer codes গুলোকে map করতে চাই। এটি করার স্বাভাবিক উপায় হলো একটি array of Strings ব্যবহার করা। যেভাবে আমরা array of primitive types তৈরি করেছিলাম ঠিক সেভাবেই আমরা array of Strings তৈরি করতে পারি:

```
String[] suits = new String[4];
```

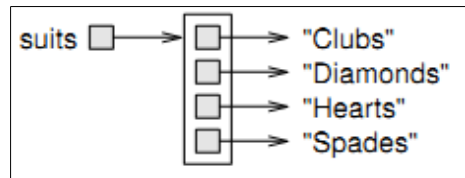
এরপর আমরা array এর উপাদান বা elements এর values নির্দিষ্ট করতে পারি।

```
suits[0] = "Clubs";
suits[1] = "Diamonds";
suits[2] = "Hearts";
suits[3] = "Spades";
```

array তৈরি করা এবং elements গুলোকে initialize করার জাভার একটি খুব সাধারণ operation যার কারণে

জাভা এর জন্য বিশেষ syntax প্রদান করে:

```
String[] suits = { "Clubs", "Diamonds", "Hearts", "Spades" };
```

এই statement আলাদা আলাদা declaration, allocation, এবং assignment এর সমতুল্য। এই array এর state diagram নিম্নরূপ:

Strings গুলো নিজেরা যেভাবে কাজ করে তারচেয়ে array এর elements, Strings এর references হিসেবে বরং সেই কাজ করে।

এখন আমাদের অন্য একটি array of Strings দরকার ranks-কে decode করার জন্য:

```
String[] ranks = { "narf", "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen",
"King" };
```

"narf" ব্যবহার করার কারণ হলো এটি array -এর zeroeth element-এর জন্য place-keeper হিসেবে কাজ করবে; যা কখনো ব্যবহার হবে না বা ব্যবহার করা যাবে না। এখানে valid ranks গুলো হলো শুধুমাত্র 1 -13। Element এর এমন অপচয় দূর করা জন্য আমরা 0 দিয়ে শুরু করতে পারি। কিন্তু আমাদের mapping অনেক বেশি বাস্তবসম্মত হবে যদি আমরা 2 কে 2 এবং 3 কে 3, ইত্যাদি, হিসেনে encode করি।

এই arrays গুলোকে ব্যবহার করে, আমরা সঠিক Strings নির্বাচন করতে পারি। এর জন্য আমাদের indices হিসেবে suit
এবং rank কে ব্যবহার করতে হবে। printCard method এ,

```
public static void printCard(Card c) {
        String[] suits = { "Clubs", "Diamonds", "Hearts", "Spades" };
        String[] ranks = { "narf", "Ace", "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack",
                "Queen", "King" };
        System.out.println(ranks[c.rank] + " of " + suits[c.suit]);
}
```

suits[c.suit] এই expression এর মানে "array নামের suits থেকে index হিসেবে ব্যবহৃত object c এর instance variable suit ব্যবহার করা এবং সঠিক string নির্বাচন করা।" এই কোডের আউটপুট হলো;

```
Card card = new Card(1, 11);
printCard(card);
```

is Jack of Diamonds.

১৩.৪ sameCard মেথড (The sameCard method)

"same" শব্দটি স্বাভাবিক ভাষায় যা বোঝায় সে জিনিসগুলোর একটি। এক্ষেত্রে আমরা যদি ভিন্ন কোন চিন্তা না আনি তাহলে তা একই থেকে যাবে এবং তখন আমরা বুঝতে পারবো আমরা যা আশা করেছিলাম তার মধ্যে তার চেয়ে বেশি আছে।

উদাহরণস্বরূপ, যদি আমি বলি "করিম এবং আমার গাড়ি একই'," এর অর্থ আমার এবং করিমের গাড়ি একই মডেল এবং একইভাবে তৈরি। কিন্তু এগুলো দুটি ভিন্ন গাড়ি। আবার আমি যদি বলি " করিম এবং আমার মা একই, " এর অর্থ করিমের মা এবং আমার মা একজন ব্যক্তি। তাই "sameness" এর ধারণা বিষয়বস্তু বা context ওপর ভিত্তি করে বিভিন্ন হতে পারে।

যখন তুমি objects নিয়ে কথা বলবে, তখনও এধরনের ambiguity বা দ্ব্যর্থতা থাকবে। উদাহরণস্বরূপ, যদি দুটি Cards একইরকম হয় তাহলেই কি তার অর্থ তারা একই ডেটা ধারণ করে অথবা তাদের rank এবং suit একই, অথবা আসলেই কি তারা একই বা Same Card object?

যদি দুটি references একই অবজেক্টকে refer করে তাহলে আমরা == operator ব্যবহার করতে পারি। উদাহরণস্বরূপ:

```
Card card1 = new Card(1, 11);
Card card2 = card1;

if (card1 == card2) {
        System.out.println("card1 and card2 are identical.");
}
```

একই অবজেক্টের references গুলো হবে অভিন্ন বা identical। অবজেক্টের references যদি একই ডেটা দিয়ে হয় তাহলে তা সমতুল্য বা equivalent।

সমতুল্যতা পরীক্ষা করার জন্য, সাধারণ উপায় হলো একটি method লেখা যার নাম হবে sameCard।

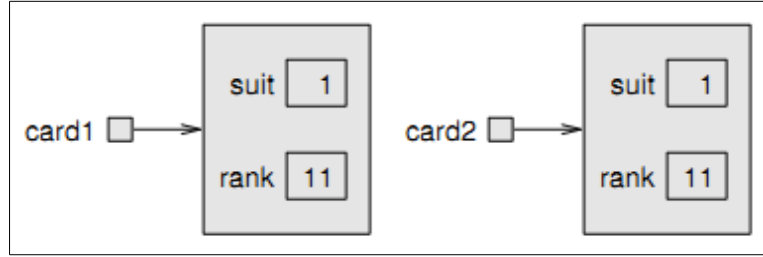
```
public static boolean sameCard(Card c1, Card c2) {
        return(c1.suit == c2.suit && c1.rank == c2.rank);
}
```

এখানে একটি উদাহরণ রয়েছে যা একই ডেটা দিয়ে দুটি অবজেক্ট তৈরি করবে এবং sameCard ব্যবহার করতে তারা সমতুল্য কিনা তা দেখার জন্য:

```
Card card1 = new Card(1, 11);
Card card2 = new Card(1, 11);

if (sameCard(card1, card2)) {
        System.out.println("card1 and card2 are equivalent.");
}
```

যদি references গুলো অভিন্ন হয় তাহলে তারা অবশ্যই সমতুল্য কিন্তু যদি তারা সমতুল্য হয় তাহলে তাদের অভিন্ন হতেই হবে এমন কোন প্রয়োজনীয়তা নেই।

এক্ষেত্রে card1 এবং card2 হলো সমতুল্য কিনুত তারা অভিন্ন নয়, তাই state diagram দেখতে হবে নিচের মতো:

যখন card1 এবং card2 অভিন্ন হবে তখন তা দেখতে কেমন হবে ?

8.10 সেকশনে আমরা বলেছিলাম, আমাদের Strings এর ওপর == operator ব্যবহার করা উচিত নয় কারণ আমরা যা চাই এটি তখন তা করে না। String (equivalence) এর বিষয়বস্তু বা contents কে তুলনা করা ছাড়াও এটি পরীক্ষা করে দেখে দুটি এর অবজেক্ট object (identity) একই বা অভিন্ন কিনা।

১৩.৫ compareCard মেথড (The compareCard method)

primitive types এর জন্য, conditional operators গুলো values তুলনা করে এবং নির্ধারণ করে কখন কোনটি অন্যটির চেয়ে বড় বা ছোট হবে। এই operators (< এবং > এবং অন্যান্যগুলো) object types এর জন্য কাজ করে না। Strings এর জন্য জাভার একটি compareTo method রয়েছে। কার্ডের জন্য আমাদের নতুন করে method লিখতে হবে, যাকে আমরা বলতে পারি compareCard। পরে, আমরা এই method কে ব্যবহার করতে পারি deck of cards বাছাই করার জন্য।

কিছু sets হলো পুরোপুরি বিন্যস্তভাবে (ordered) সাজানো, যার অর্থ তুমি যে কোন দুটি উপাদানের তুলনা করতে পারো এবং বলতে পারো কোনটি বড়। Integers এবং floating-point numbers গুলো সম্পূর্ণরূপে বিন্যস্তভাবে থাকে। তবে কিছু অবিন্যস্ত (unordered) sets থাকে এর অর্থ এক্ষেত্রে আমাদের এমন কোন অর্থপূর্ণ উপায় থাকে না যেগুলো দিয়ে আমরা বের করতে পারি কোন উপাদান কোনটির চেয়ে বড়। যেমন বিভিন্ন ফলগুলো হলো অবিন্যস্ত, এ কারণেই আমরা আপেল আর কমলার তুলনা করতে পারি না। জাভাতে boolean type হলো অবিন্যস্ত। একইভাবে আমরা বলতে পারি না সত্য মিথ্যার চেয়ে বড়।

playing cards খেলার set হলো আংশিকভাবে বিন্যস্ত, এর অর্থ আমরা কখনো কখনো cards গুলো তুলনা করতে পারি আবার কখনো কখনো তুলনা করতে পারি না। উদাহরণস্বরূপ, আমরা জানি 3 of Clubs হলো 2 of Clubs এর চেয়ে বড় এবং 3 of Diamonds হলো 3 of Clubs এর চেয়ে বড়। কিন্তু কোনটা ভালো, 3 of Clubs অথবা 2 of Diamonds এর মধ্যে কোনটা ভালো? একটির রয়েছে উচ্চতর rank, কিন্তু অন্যটির রয়েছে উচ্চতর suit।

cards গুলোকে তুলনাযোগ্য করার জন্য, আমাদের সিদ্ধান্ত নিতে হবে rank অথবা suit এর মধ্যে কোনটি বেশি

গুরুত্বপূর্ণ। এক্ষেত্রে ইচ্ছা আমাদের অবাধ, কিন্তু যখন আমরা নতুন deck of cards কিনবো, তখন এটি sorted হবে সব Clubs গুলো সহ, যার মধ্যে রয়েছে সব Diamonds এবং অন্যান্যগুলো। তাই চলো আমরা বলার চেষ্টা করি ঐ suit হলো বেশি গুরুত্বপূর্ণ।

এই সিদ্ধান্তের আলোকে আমরা compareCard লিখতে পারি। এটি প্যারামিটার হিসেবে দুটি Cards নিবে এবং যদি প্রথম কার্ড বিজয়ী হয় তাহলে রিটার্ন করবে 1 এবং যদি দ্বিতীয় কার্ড বিজয়ী হয় তাহলে তা -1 রিটার্ন করবে এবং 0 রিটার্ন করবে যদি তারা সমতুল্য হয়।

প্রথম আমরা তুলনা করি suits গুলো:

```
if (c1.suit > c2.suit) return 1;
if (c1.suit < c2.suit) return -1;
```

যদি statement-টি true না হয়, তাহলে suits গুলো অবশ্যই সমান হবে, এবং আমাদের ranks গুলো তুলনা করতে হবে:

```
if (c1.rank > c2.rank) return 1;
if (c1.rank < c2.rank) return -1;
```

যদি এই দুটির কোনটিই true না হয়, তাহলে ranks গুলো অবশ্যই সমান হতে হবে, তাহলে আমরা রিটার্ন করতে পারবো 0।

১৩.৬ অ্যারেস অব কার্ড (Arrays of cards)

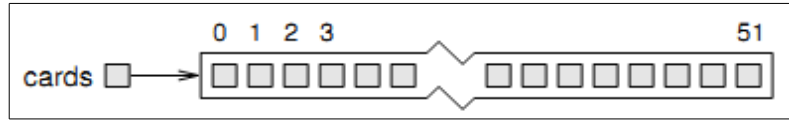
এখন পর্যন্ত আমরা composition (the ability to combine language features in a variety of arrangements) এর বিভিন্ন উদাহরণ দেখলাম। প্রথম উদাহরণে আমরা একটি expression এর অংশ হিসেবে method invocation করা দেখলাম। অন্য উদাহরণটি হলো nested structure of statements: তুমি একটি while loop এর ভিতরে একটি if statement রাখতে পারো অথবা একটি if statement ভিতরে অন্য একটি if statement-ও রাখতে পারো ইত্যাদি।

এই নিয়ম দেখার পরে এবং arrays ও objects বিষয়ে জানার পরে, আমাদের আসলে অবাক হবার অবসর নেই যদি আমরা arrays of objects তৈরি করতে পারি। একইভাবে আমরা arrays কে instance variables হিসেবে প্রকাশ করতেও পারি; আমরা এমন arrays ও তৈরি করতে পারি যা অন্য arrays কেও ধারণ করতে পারি; আমরা এমন অবজেক্ট define করতে পারি যা অন্য অবজেক্টকে ধারণ করতে পারি এবং এই রকম আরও অনেক কিছু করতে পারি।

এর পরের দুটি অধ্যায়ে আমরা বিভিন্ন উদাহরণ দেখবো যা Card objects এর combination ব্যবহার করে করা হবে।

এই উদাহরণটি তৈরি করবে array of 52 cards:

```
Card[] cards = new Card[52];
```

এখানে অবজেক্টের state diagram হলো:

Array ধারণ করে অবজেক্টের references; এটি সাধারণত Card objects গুলোকে ধারণ করে না। উপাদানগুলোকে null হিসেবে initialized করা হয়। সাধারণ নিয়মেই আমরা array এর উপাদানগুলোকে access করতে পারি:

```
if (cards[0] == null) {
        System.out.println("No cards yet!");
}
```

কিন্তু যদি আমরা non-existent Cards দিয়ে instance variables কে access করতে চাই, তাহএল আমরা পাবো NullPointerException।

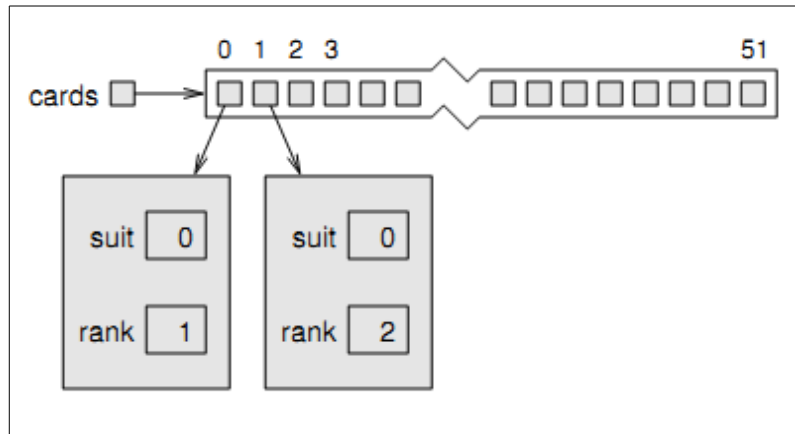
```
cards[0].rank;                    // NullPointerException
```

কিন্তু এটি হলো deck এর "zeroeth" rank এর card access করার সঠিক syntax। এটি হলো একটি array এর উপাদান access করার এবং একটি অবজেক্টের instance variable access করার composition, combining syntax।

Deck সহ Card objects কে populate করার সহজ উপায় হলো nested for loops লেখা (অর্থাৎ, একটির ভিতরে আর একটি loop):

```
int index = 0;
for (int suit = 0; suit <= 3; suit++) {
        for (int rank = 1; rank <= 13; rank++) {
                cards[index] = new Card(suit, rank);
                index++;
        }
}
```

Outer loop টি 0 থেকে 3 suits উল্লেখ করে। আর প্রত্যেক suit এর জন্য, inner loop টি ranks এর 1 থেকে 13 উল্লেখ করে। যেহেতু outer loop টি ৪ বার চলে এবং inner loop টি 13 বার চলে, তাই body এর execution হবে ৫২ বার।

আমি পরবর্তী কার্ডের deck এর track রাখার জন্য অর্থাৎ কোথায় কার্ডটি যাবে তা বোঝার জন্য index ব্যবহার করেছি। নিচের state diagram টি আমাদের সামনে তুলে ধরবে প্রথম দুটি কার্ড allocate হবার পর deck-গুলো কেমন দেখাবে:

১৩.৭ printDeck মেথড (The printDeck method)

যখন আমরা arrays নিয়ে কাজ করি, তখন একটি method নিয়ে কাজ করা সুবিধাজনক যা contents প্রিন্ট করবে। আমরা বিভিন্ন সময়ে আমরা array খোঁজার (traversing) বিভিন্ন pattern দেখি, সুতরাং নিচের method হবে নিম্নরূপ:

```
public static void printDeck(Card[] cards) {
        for (int i = 0; i < cards.length; i++) {
                printCard(cards[i]);
        }
}
```

যেহেতু cards এর জন্য আমরা type Card[] ব্যবহার করি, কার্ডের একটি উপাদান হিসেবে আমরা পাই type Card। তাই নিয়মানুসারে printCard এর legal argument হলো cards[i]।

১৩ .৮ খোঁজা (Searching)

পরের যে method টি আমরা লিখবো তা হলো findCard, যা কার্ডের array (array of Cards) এর মধ্যে থেকে খুঁজে বের করে নির্দিষ্ট কার্ড এর মধ্যে রয়েছে কিনা। এই method আমাদেরএকটি সুযোগ দেবে দুটি algorithms demonstrate করার। আর এই algorithms দুটি হলো: linear search এবং bisection search।

Linear search অনেকটা সুস্পষ্ট; আমরা deck-কে traverse এবং compare করতে পারি প্রত্যেকটি কার্ডের সাথে যে কার্ডটি আমরা খুঁজছি তার জন্য। যদি আমরা একে খুঁজে পাই তাহলে যেখানে কার্ড রয়েছে সেখানে আমরা index রিটার্ন করি। আর যদি deck তা না থাকে তাহলে আমরা রিটার্ন করবো -1।

```
public static int findCard(Card[] cards, Card card) {
        for (int i = 0; i< cards.length; i++) {
                if (sameCard(cards[i], card)) {
                        return i;
                }
```

```
        }
        return -1;
    }
```

findCard এর arguments হলো card এবং cards। এটা হয়তো একটু বিসদৃশ লাগবে যদি আমরা type এর নামানুসারে বা একই নামে variable নেই (card variable এর type যদি হয় Card)। আমরা কিন্তু এক্ষেত্রে পার্থক্যটি বলতে পারি, variable শুরু হয় lower-case letter বা ছোট হাতের অক্ষরে আর type শুরু হয় বড় হাতের অক্ষরে।

method টি যত তাড়াতাড়ি কার্ডটি খুঁজে পাবে তত তাড়াতাড়ি সে তা রিটার্ন করবে। এর অর্থ হলো আমাদের পুরো deck-কে traverse করার দরকার নেই যদি আমরা যে কার্ডটি খুঁজছি তা খঁজে পাই। যদি আমরা লুপটি (loop) শেষ করতে চাই তাহলে আমাদের জানতে হবে কার্ডটি deck এর মধ্যে নেই।

যদি কার্ডটি deck এর মধ্যে ক্রমানুসারে না থাকে, তাহলে এর চেয়ে তাড়াতাড়ি খুঁজে বের করার অন্য কোন উপায় নেই। আমাদের প্রত্যেকটি কার্ড দেখতে হবে কারণ তা না হলে আমরা নির্দিষ্ট কার্ডটি যথাস্থানে পাবো না।

কিন্তু যখন আমরা শব্দকোষে কোন শব্দ খুঁজে বের করতে চাইবো, তখন কিন্তু আমরা প্রত্যেকটি শব্দ ধরে ধরে খুঁজি না। এক্ষেত্রে শব্দগুলো সাজানো থাকে alphabetical order-এ। ফলশ্রুতিতে, আমরা bisection search এর মতোই একটি algorithm ব্যবহার করি:

১। মাঝের কোন জায়গা থেকে শুরু করতে হবে।
২। পৃষ্ঠার কোন একটি শব্দ পছন্দ করতে হবে এবং যে শব্দটি আমরা খুঁজছি তার সাথে মিলাতে হবে।
৩। যদি আমরা যে শব্দটি খুঁজছি তা পেয়ে যাই তাহলে আমাদের থামতে হবে।
৪। যদি যে শব্দটি খোঁজা হচ্ছে তা প্রাপ্ত শব্দের পরে শব্দকোষে যে কোন অংশে থাকে তাহলে দ্বিতীয় ধাপে যেতে হবে।
৫। যদি যে শব্দটি খোঁজা হচ্ছে তা প্রাপ্ত শব্দের আগে শব্দকোষে যে কোন অংশে থাকে তাহলে দ্বিতীয় ধাপে যেতে হবে।

যদি আমরা শব্দ খোঁজার সময় এমন একটি পয়েন্টে যাই যেখানে দুটি সংযুক্ত শব্দ রয়েছে পৃষ্ঠায় এবং আমাদের প্রয়োজনীয় শব্দটি তাদের মধ্যকার, তাহলে আমরা ধরে নিতে পারি আমরা যে শব্দটি খুঁজছি তা শব্দকোষে নেই।

আবার আমরা আমদের deck of cards এ ফিরে যাই, যদি আমরা জানি কার্ডগুলো ক্রামুসারে সাজানো রয়েছে তাহলে আমরা তুলনামূলকভাবে দ্রুত findCard এর ভার্সন লিখতে পারবো। একটি bisection search লেখার সবচেয়ে ভালো উপায় হচ্ছে recursive method, কারণ bisection হলো প্রকৃত পক্ষে recursive।

method লেখার trick-কে বলে findBisect, এটি দুটি নির্দেশক (indices) প্যারামিটার হিসেবে নিয়ে কাজ করে, low এবং high, নির্দেশ করে segment of the array, যা খুঁজে বের করতে হবে (low এবং high সহ)।

১। array খোঁজার জন্য, low এবং high এর মধ্যে একটি index পছন্দ করতে হবে। (আমরা একে বলতে পারি mid) এবং তুলনা করতে পারি আমরা যে কার্ড খুঁজছি তার সাথে।
২। যদি তমি এটি খুঁজে পাও, থামো।
৩। যদি mid এর কার্ড তোমার কার্ডের চেয়ে বড় হয়, তাহলে low হতে mid-1 পর্যন্ত খুঁজতে হবে।
৪। যদি mid এর কার্ড তোমার কার্ডের চেয়ে ছোট হয়, তাহলে mid+1 হতে high পর্যন্ত খুঁজতে হবে।

তৃতীয় এবং চতুর্থ ধাপ সন্দেহজনকভাবে (suspiciously) দেখতে অনেকটা recursive invocations এর মতো। এখানে এটি দেখতে যেমন তা যদি আমরা জাভা কোডে translated করি তাহলে হবে:

```
public static int findBisect(Card[] cards, Card card, int low, int high) {
        // TODO: need a base case

        int mid = (high + low) / 2;
        int comp = compareCard(cards[mid], card);

        if (comp == 0) {
                return mid;
        } else if (comp > 0) {
                return findBisect(cards, card, low, mid-1);
        } else {
                return findBisect(cards, card, mid+1, high);
        }
}
```

এই কোড bisection search এর kernel ধারণ করে। তারপরও এখানে এখনো একটি গুরুত্বপূর্ণ অংশ বাকি রয়েছে। যার কারণে আমাদের TODO comment সংযুক্ত করতে হয়েছে। যেহেতু, method recurses সবসময়ের জন্য লেখা হচ্ছে, যদি কার্ডটি deck এ না থেকে থাকে, তাহলে আমাদের একটি base case এর দরকার এই অবস্থাটি নিয়ন্ত্রণ করার জন্য।

যদি high, low এর চেয়ে ছোট হয়, এবং তাদের মধ্যে আর কোন কার্ড না থেকে থাকে, তাহলে আমাদের সমাপ্তি ঘোষণা করতে হবে, কার্ডটি deck-এর মধ্যে নেই। যদি আমরা এই case-টি নিয়ন্ত্রণ করতে পারি, তাহলে method-টি ভালোভাবে কাজ করবে:

```
public static int findBisect(Card[] cards, Card card, int low, int high)
        System.out.println(low + ", " + high);
        if (high < low) return -1;
        int mid = (high + low) / 2;
        int comp = compareCard(cards[mid], card);

        if (comp == 0) {
                return mid;
        } else if (comp > 0) {
                return findBisect(cards, card, low, mid-1);
        } else {
                return findBisect(cards, card, mid+1, high);
        }
}
```

আমি একটি print statement সংযুক্ত করতে চাচ্ছি যেন আমি sequence of recursive invocations

অনুসরণ করতে পারি। আমি নিচের কোডের সাহায্যে চেষ্টা করছি:

```
Card card1 = new Card(1, 11);
System.out.println(findBisect(cards, card1, 0, 51));
```

এবং আমি পাবো নিচের output:

```
0, 51
0, 24
13, 24
19, 24
22, 24
23
```

এরপর আমি একটি কার্ড তৈরি করবো যা deck-এ নেই (the 15 of Diamonds), এবং চেষ্টা করবো একে খুঁজে বের করার। এবং আমি পাবো নিচের আউটপুট:

```
0, 51
0, 24
13, 24
13, 17
13, 14
13, 12
-1
```

এই পরীক্ষাই প্রমাণ করে না যে এই প্রোগ্রামটি সঠিক। এমনকি, কোন amount এর পরীক্ষাই প্রমাণ করতে পারবে না এই প্রোগ্রামটি সঠিক। অন্যদিকে, কিছু cases দেখে এবং তাদের code পরীক্ষা করে, আমরা হয়তো আমাদের সামনে প্রমাণ উপস্থাপন করতে পারবো।

recursive invocations এর সংখ্যা হলো সাধারণত 6 অথবা 7, সুতরাং আমরা compareCard, invoke করতে পারবো শুধুমাত্র 6 অথবা 7 বার, আবার আমরা যদি linear search ব্যবহার করতাম তাহলে আমরা compared করতে পারতাম 52 বার।সাধারণভাবে, bisection, linear search এরচেয়ে অনেক দ্রুততর, এবং এমনকি অনেক বড় arrays এর জন্যও এটি প্রযোজ্য।

recursive programs এর দুটি সাধারণ ত্রুটি হলো base case -কে সংযুক্ত করতে ভুলে যাওয়া এবং recursive call লেখা যেন base case কখনো reached না হয়। আবার innite recursion এর জন্যও ত্রুটি হতে পারে, যা আমাদের জন্যএকটি StackOverflowException নিক্ষেপ করতে পারে। (একটি recursive method এর জন্য একটি stack diagram এর চিন্তা করো যা কখনোই শেষ হবে না।)

১৩.৯ ... ...(Decks and subdecks)

এখানে রয়েছে findBisect এর একটি prototype (সেকশন ৮.৫ দেখো):

```
public static int findBisect(Card[] deck, Card card, int low, int high)
```

আমরা কার্ড নিয়ে চিন্তা করতে পারি, low, এবং high একটি একক parameter যা হলো subdeck এর একটি species। এই ধরনের চিন্তা খুব সাধারণ এবং মাঝে মাঝে এটি referred করে একটি abstract parameter হিসেবে। এখানে "abstract" হিসেবে বোঝানো হচ্ছে এমন কিছু যা আক্ষরিক অর্থে program text এর অংশ নয়, কিন্তু যা higher level এ function of
the program বর্ণনা করে।

উদাহরণস্বরূপ, যখন আমরা একটি method, invoke করি এবং একটি array, pass করি এবং bounds হয় low এবং high। এখানে এমন কিছু নেই যা invoked method এর bounds-এর বাইরের array এর বিভিন্ন অংশ access করা প্রতিরোধ করে।সুতরাং আমরা আক্ষরিকভাবে deck-এ কোন subset পাঠাচ্ছি না। আমরা মূলত পুরো deck-ই পাঠাচ্ছি। কিন্তু যতক্ষণ গ্রহণকারী নিয়মানুসারে চলে, এটিকে একটি subdeck হিসেবে abstractly আমরা চিন্তা করতে পারি।

এই ধরনের চিন্তা, যেখানে একটি প্রোগ্রাম কোন অর্থ বহন করে না, এমনকি literally encoded করা হলেও না; এটি কম্পিউটার বিজ্ঞানী হিসেবে চিন্তা করার একটি গুরুত্বপূর্ণ অংশ। "abstract" শব্দটি আমরা প্রায়ই ব্যবহার করি এবং অনেক ক্ষেত্রেই এটি এর অর্থ হারায়। তাস্বত্ত্বেও, কম্পিউটার বিজ্ঞানে (অন্য অনেক ক্ষেত্রে) abstraction একটি কেন্দ্রিয় ধারণা।
"abstraction" এর আরো ভালো সাধারণ সংজ্ঞা হলো "জটিল বা complex system এর modeling প্রক্রিয়া, যা সাধারণভাবে বর্ণনা করে অপ্রয়োজনীয় বিবিধগুলো যখন সে capture করে relevant behavior।"

১৩.১০ শব্দকোষ (Glossary)

encode: দুই set values-এর মধ্যে mapping তৈরি করে, এক set values-কে represent করা, অন্য এক set values এর সাহায্যে

identity: references এর সমতা। দুটি references যা memory-এর একই object নির্দেশ করে।

equivalence: values এর সমতা। দুটি references যা এমন অবজেক্ট নির্দেশ করে যা একই ডেটা ধারণ করে।

abstract parameter: এক set parameters যা একটি একক parameter হিসেবে একসাথে কাজ করে।

abstraction: higher level-এ প্রোগ্রামকে interprete করার একটি প্রক্রিয়া (অথবা অন্য কিছু) যা literally উপস্থাপন করা হয় code দ্বারা।

১৩.১১ অনুশীলনী (Exercises)

অনুশীলনী ১৩.১

সেকশন ১৩.৫ এর কোড একটি method-এর মাধ্যমে Encapsulate করো। এরপর একে modify করো যেন aces-গুলো Kings এরচেয়ে উঁচু rank এ থাকে।

অনুশীলনী ১৩.২

সেকশন১৩.৬ এর deck-building কোড makeDeck নামক method দ্বারা encapsulate করতে হবে যা কোন প্যারামিটার নেবে না এবং রিটার্ন করবে একটি পুরো populated array of Cards.

অনুশীলনী ১৩.৩

Blackjack এ গেইম-এর object এক গুচ্ছ কার্ড গ্রহণ করে যার score 21। hand এর score সবগুলো কার্ডের স্কোরের (scores) -এর যোগফল। একটি aces এর score হলো 1, সবগুলো face cards এর স্কোর হলো ten এবং অন্য সব কার্ডের score হলো rank এর মতো বা সমান। উদাহরণ: hand (Ace, 10, Jack, 3) মোট score হলো 1 + 10 + 10 + 3 = 24।

handScore নামে একটি method লিখতে হবে যা একটি array of cards, argument হিসেবে নেবে এবং রিটার্ন করবে মোট score।

অনুশীলনী ১৩.৪

Poker খেলায় "flush" হলো একটি hand যা ধারণ করে পাঁচ বা তার বেশি কার্ড একই suit-এ। একটি hand যে কোন সংখ্যক কার্ড নিতে পারে।

১। suitHist নামে একটি নতুন method তৈরি করো যা একটি array of Cards প্যারামিটার হিসেবে গ্রহণ করে এবং hand এর এর মধ্যকার suits এর histogram রিটার্ন করে। তোমার সমাধান array-কে মাত্র একবারই traverse করবে।
২। hasFlush নামে একটি method লিখতে হবে, যা array of Cards-কে প্যারামিটার হিসেবে নিয়ে কাজ করবে এবং hand
যদি flush ধারণ করে তাহলে true রিটার্ন করবে এবং অন্যথায় false রিটার্ন করবে।

অনুশীলনী ১৩.৫

যদি আমরা কার্ডের খেলাকে স্ক্রিনে দেখতে পারি বা display করতে পারি, তাহলে তা আরও অনেক বেশি মজার। যদি তোমরা Appendix A এর graphics examples না খেলে থাকো তাহলে, তুমি হয়তো এখন সেটা খেলতে চাইবে।

তাহলে ডাউনলোড করে নাও এই লিংক থেকে http://thinkapjava.com/code/CardTable.java এবং http://thinkapjava.com/code/cardset.zip লিংক থেকে।

cardset.zip -কে Unzip করতে হবে এবং CardTable.java run করতে হবে। তুমি সবুজ রঙ-এর টেবিলে ("table") এক প্যাকেট কার্ড শোয়ানো দেখতে পাবে।

তুমি এই class ব্যবহার করে তোমার কার্ড গেইম খেলার শুরু স্থান হিসেবে একে বিবেচনা করতে পারো।

অধ্যায় ১৪
অ্যারের অবজেক্টসমূহ

সতর্কবার্তা: এই অধ্যায়ে আমরা অবজেক্ট-ওরিয়েন্টেড প্রোগ্রমিং এ আরেকটি পদক্ষেপ নিব, কিন্তু আমরা এখনও জায়গা মত আসি নি। তাই অনেক উদাহরণ রীতিসিদ্ধ নয়, আর তারা ভাল জাভা প্রোগ্রামও নয়। এই পরিবর্তনশীল ফর্ম তোমাকে শিখতে সাহায্য করবে (আমি আশা করছি), তবে এভাবে কোড লিখবে না।

তুমি এই অধ্যায়ের কোডটি http://thinkapjava.com/code/Card2.java থেকে ডাউনলোড করতে পার।

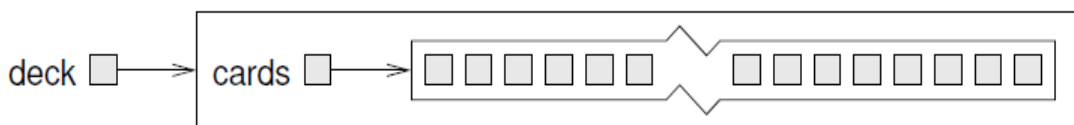
১৪.১ ডেক ক্লাস

পূর্ববর্তী অধ্যায়ে, আমরা একটি অ্যারের অবজেক্টের অ্যারের সাথে কাজ করেছি, কিন্তু আমি আরও উল্লেখ করেছি যে এটি একটি ইনসট্যান্স ভ্যারিয়েবল এর মত যা একটি অবজেক্ট ধারণ করে পাওয়া সম্ভব। এই অধ্যায়ে আমরা একটি ডেক অবজেক্ট তৈরি করতে পারি যা Cards এর একটি অ্যারে ধারণ করে।

ক্লাস ডেফিনেশন এরকম হয়:

```
class Deck {
        Card[] cards;

        public Deck(int n) {
                this.cards = new Card[n];
        }
}
```

কন্সট্রাকটর কার্ডের একটি অ্যারে ইনসট্যান্স ভ্যারিয়েবল এর সাথে শুরু হয়, কিন্তু কোন কার্ড তৈরি করে না। এখানে কার্ডবিহীন একটি ডেক যেরকম তার মত একটি স্টেট ডায়াগ্রাম প্রদর্শিত হয়েছে:

এখানে কোন আর্গুমেন্ট কন্সট্রাকটর নেই যা ৫২-কার্ড ডেক তৈরি করে এবং এটি কার্ডের সাথে পূর্ণ করে:

```
public Deck() {
        this.cards = new Card[52];
        int index = 0;
        for (int suit = 0; suit <= 3; suit++) {
                for (int rank = 1; rank <= 13; rank++) {
                        cards[index] = new Card(suit, rank);
                        index++;
                }
```

```
        }
    }
```

এই মেথডটি makeDeck এর মত; আমরা শুধুমাত্র সিনট্যাক্স পরিবর্তন করেছি এটিকে একটি কন্সট্রাকটর তৈরির জন্য।
এটিকে ডাকার জন্য আমরা new ব্যবহার করি:

```
    Deck deck = new Deck();
```

ডেকের ক্লাস ডেফিনেশনে ডেকের সাথে সম্পর্কযুক্ত মেথডটি এখন অর্থপূর্ণ। পূর্বে মেথডে আমরা যা লিখেছি, তার মধ্যে একটি অনস্বীকার্য ক্যান্ডিডেট হল printDeck (সেকশন ১৩.৭)। এখানে ডেক এর সাথে কাজ করতে এটি যেরকম দেখতে হবে তা পুনরায় লেখা হল:

```
    public static void printDeck(Deck deck) {
        for (int i = 0; i < deck.cards.length; i++) {
            Card.printCard(deck.cards[i]);
        }
    }
```

প্যারামিটারের ধরনের Card [] থেকে ডেক হল একটি পরিবর্তন।

দ্বিতীয় পরিবর্তনটি হল যে আমরা অ্যারে এর length এর জন্য আর deck.length ব্যবহার করব না, কারণ ডেক এখন একটি অ্যারে নয়, এটি এখন একটি ডেক অবজেক্ট। এটি একটি অ্যারে ধারণ করে, কিন্তু এটি একটি অ্যারে নয়। তাই আমাদের ডেক অবজেক্ট থেকে অ্যারে বের করার জন্য deck.cards.length লিখতে হবে এবং তাহলে আমরা অ্যারে এর ব্যাপ্তি পাব।

একই কারণে, আমাদের শুধুমাত্র deck [i] এর বদলে একটি অ্যারের উপাদান একসেস করার জন্য deck.cards [i] ব্যবহার করতে হবে।

printCard এর আহ্বানের শেষের পরিবর্তনটি বিশদভাবে বলতে গেলে আমরা বলতে পারি printCard টি Card class এ সংজ্ঞায়িত হয়েছে।

১৪.২ শাফলিং (অদলবদল)

অধিকাংশ কার্ড গেমে তোমকে ডেক শাফল করার প্রয়োজন হবে, যা কার্ডকে র‍্যান্ডম অর্ডারে সাজাবে। সেকশন ১২.৬ এ আমরা দেখেছি কিভাবে র‍্যান্ডম নম্বর তৈরি করতে হয়, কিন্তু এটি সুস্পষ্ট ছিল না যে কিভাবে একটি ডেক শাফল করতে হয়।

একটি সম্ভাবনা হল যে humans shuffle উপায়ে মডেল করা, যা সাধারণত করা হয় ডেকটি দুইভাগে বিভক্ত করে এবং এরপর প্রতিটি ডেক থেকে পর্যায়ান্বিত করে নির্বাচন করা হয়। যেহেতু মানুষ সাধারণত সঠিকভাবে শাফল করতে পারে না, ৭ বার পুনরাবৃত্তির পর ডেক এর অর্ডার মোটামুটি ভাল র‍্যান্ডম অনুসারে হয়। কিন্তু একটি কম্পিউটার প্রোগ্রামের বিরক্তিকর বৈশিষ্ট্য রয়েছে যা প্রতিবার একটি সঠিক শাফল করে, যা সত্যিই খুব র‍্যান্ডম নয়। আসল ঘটনা হল, ৮ টি সঠিক শাফল এর পর, তুমি যেভাবে শুরু করেছিলে সেভাবে তুমি ডেকটি ফেরত পাবে।

আরো তথ্য জানার জন্য, http://en.wikipedia.org/wiki/Faro_shuffle দেখ।

একটি শাফলিং অ্যালগরিদম একই সময়ে ডেকটি একটি কার্ড অতিক্রম করে, এবং প্রতি ইন্টারেকশনে দুটি কার্ড নির্বাচন করে এবং সোয়াপ করে।

এখানে একটি আউটলাইন আছে যে কিভাবে অ্যালগরিদম কাজ করে। এই প্রোগ্রামটি স্কেচ করার জন্য, আমি জাভা স্টেটমেন্টের এবং ইংরেজি ওয়ার্ডের একটি কম্বিনেশন ব্যবহার করি যাকে মাঝে মাঝে pseudocode বলা হয়:

```
for (int i = 0; i < deck.cards.length; i++) {
    // choose a number between i and deck.cards.length-1
    // swap the ith card and the randomly-chosen card
}
```

pseudocode এর একটি সুন্দর ব্যাপার হল যে এটি কোন কোন সময়ে তোমার কোন মেথডটি দরকার তা পরিষ্কার করে দেয়। এই জন্য, আমাদের randomInt এর মত কিছু দরকার; যা low এবং high এর মধ্যে একটি র‍্যান্ডম ইন্টিজার পছন্দ করবে, এবং swapCards করবে যা দুটি সূচক নেবে এবং চিহ্নিত অবস্থানসমূহে কার্ড সুইচ করবে।

এই প্রক্রিয়ায়- প্রথমে pseudocode লিখতে হয় এবং পরবর্তীতে এটি কাজ করার জন্য মেথড লিখতে হয়- যাকে বলা হয় top-down development (দেখুন http://en.wikipedia.org/wiki/Top-down_and_bottom-up_design)।

১৪.৩ সর্টিং

আমরা ডেকটিতে বিশৃঙ্খল অবস্থার সৃষ্টি করে ফেলেছি, আমাদের তা প্রতিহত করতে একটি উপায় প্রয়োজন। সেখানে সর্টিং এর জন্য একটি অ্যালগরিদম রয়েছে যা শাফলিং এর জন্য অ্যালগরিদমের হাস্যকর অনুরূপ। এটিকে selection sort বলা হয় কারণ এটি অ্যারে ট্রাভারসিং করার মাধ্যমে কাজ করে এবং প্রতিবার নিম্ন অবশিষ্ট কার্ড নির্বাচন করে।

প্রথম মিথস্ক্রিয়ার সময় আমরা সর্বনিম্ন কার্ডটি খুঁজে বের করেছি এবং 0 তম অবস্থানের কার্ডের সাথে এটি অদলবদল করেছি। i তম এর সময়, আমরা i এর সর্বনিম্ন কার্ড খুঁজে বের করেছি এবং i কার্ডের সাথে এটি অদলবদল করেছি।

সিলেকশন সর্টের প্রক্রিয়া এখনে দেয়া হল:

```
for (int i = 0; i < deck.cards.length; i++) {
    // find the lowest card at or to the right of i
    // swap the ith card and the lowest card
}
```

pseudocode টি আবার helper methods ডিজাইনের সাথে সহায়তা করে। এই ব্যাপারে আমরা পুনরায় swapCards ব্যবহার করতে পারি, তাই আমাদের শুধুমাত্র একটি নতুন দরকার, যাকে indexLowestCard বলা হয়, যা কার্ডের একটি অ্যারে এবং একটি ইনডেক্স নেয় যেখানে এটি খোঁজা শুরু করবে।

১৪.৪ সাবডেকসমূহ

কিভাবে আমরা পূর্ণ ডেকের একটি হ্যান্ড অথবা অন্য কোন সাবসেট উপস্থাপন করতে পারি? একটি সম্ভাবনা হল Hand নামক একটি ক্লাস তৈরি করা, যা ডেকটিকে সম্প্রসারণ করতে পারে। অন্য সম্ভাবনা আমি যা প্রমান করেছি তা হল ৫২ কার্ডের চেয়ে কমে একটি হ্যান্ডের সহিত একটি ডেক অবজেক্ট এর সাথে উপস্থাপন করা।

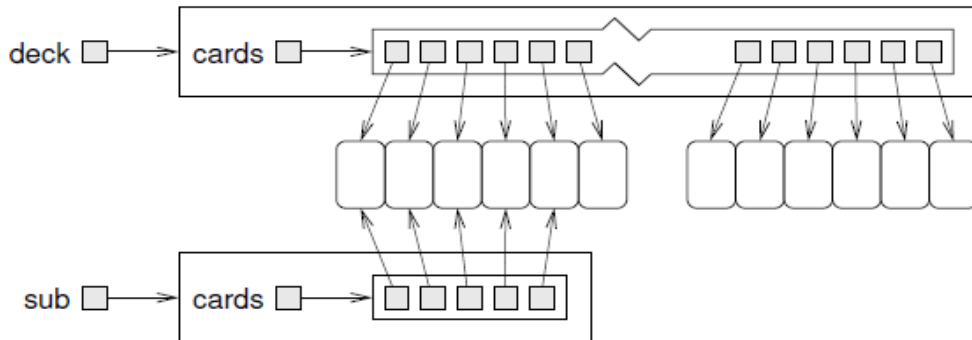
আমারা হয়ত সাবডেক নামক একটি মেথড চাইতে পারি, যা একটি ডেক নেয় এবং সূচকসমূহের একটি ব্যাপ্তি নেয়, এবং একটি নতুন ডেক ফেরত দেয় যা কার্ডেসমূহের নির্দিষ্ট সাবসেট ধারণ করে:

```
public static Deck subdeck(Deck deck, int low, int high) {
        Deck sub = new Deck(high-low+1);

        for (int i = 0; i<sub.cards.length; i++) {
                sub.cards[i] = deck.cards[low+i];
        }
        return sub;
}
```

সাবডেকের দৈর্ঘ্য হল -low+1 কারণ উভয়ের সাথেই নিম্ন কার্ড এবং উচ্চ কার্ড সংযুক্ত রয়েছে। এই ধরনের গণনা বিভ্রান্তিকর হতে পারে, এবং "off-by-one" ত্রুটি দ্বারা পরিচালিত হতে পারে। একটি চিত্র অংকন হতে পারে তাদের এড়িয়ে যাওয়ার একটি উত্তম পথ।

কারন আমরা new এর সাথে একটি আর্গুমেন্ট প্রদান করি, কন্ট্রাকটর যা invoked পায় তা প্রথমটি হতে পারে, যা শুধু অ্যারে বন্টন করে এবং কোন কার্ড বন্টন করে না। for loop এর অভ্যন্তরে, ডেক থেকে রেফারেন্স এর অনুলিপির সাথে সাবডেক পূর্ণ হয়।

নিম্ন লিখিত ডায়াগ্রামটি হল একটি সাবডেক এর স্টেট ডায়াগ্রাম যা low=3 এবং high=7 প্যারামিটারে তৈরি হয়েছে। ফলাফল হল ৫ টি কার্ডের সাথে একটি হ্যান্ড যা অরিজিনাল ডেক এর সাথে শেয়ার করা হয়েছে, মানে হচ্ছে তারা অ্যালিয়াস হয়েছে।

অ্যালিয়াস সাধারণত একটি ভাল ধারণা নয়, কারণ একটি সাবডেকের পরিবর্তন অন্যান্যদের উপর প্রভাব ফেলে, যা তুমি রিয়েল কার্ড এবং ডেক থেকে আশা করনি। কিন্তু কার্ডগুলো যদি অপরিবর্তনীয় হয়, অ্যালিয়াসকরণ তখন কম ঝুঁকিপূর্ণ হয়। এই ঘটনায়, সেখানে সাধারণত একটি কার্ডের rank অথবা suit এর পরিবর্তন হওয়ার কোন কারন

নেই। পরিবর্তে আমরা প্রতিটি কার্ড একবার তৈরি করতে পারি এবং অপরিবর্তনীয় অবজেক্ট হিসাবে এটি ব্যবহার করতে পারি।

১৪.৫ শাফলিং এবং ডিলিং

সেকশন ১৪.২ এ আমি একটি pseudocode লিখেছি একটি শাফলিং অ্যালগরিদম এর জন্য। ধরে নেয়া যায় যে আমাদের suffleDeck নামক একটি মেথড আছে যা একটি আর্গুমেন্ট হিসাবে একটি ডেক নেয় এবং এটি শাফল করে, আমরা এটি ব্যবহার করতে পারি হ্যান্ড নিয়ন্ত্রন করার জন্য:

```
Deck deck = new Deck();
shuffleDeck(deck);
Deck hand1 = subdeck(deck, 0, 4);
Deck hand2 = subdeck(deck, 5, 9);
Deck pack = subdeck(deck, 10, 51);
```

এই কোড একটি হ্যান্ডে ৫টি কার্ড রাখে, অন্যটিতে রাখে পরের ৫টি কার্ড, এবং পরবর্তীগুলো রাখে প্যাক এর মধ্যে।

যখন আপনি ডিলিং এর কথা ভাববেন, আপনি কি চিন্তা করেছেন যে আমরা round-robin স্টাইলে প্রতিটি প্লেয়ারকে একটি কার্ড দিতে পারি কিনা যা প্রতিটি কার্ড গেমে প্রচলিত? আমি সেগুলোর বিষয়ে চিন্তা করে দেখেছি যে এটি একটি কম্পিউটার প্রোগ্রামের জন্য অপ্রয়োজনীয়। round-robin convention হল ত্রুটিপূর্ণ শাফলিং কে প্রশমিত করা এবং ডিলারের জন্য বিষয়টি আরও কঠিন করে তোলা যাতে সে চিট করতে না পারে। এটির কোনটিই কম্পিউটারের জন্য কোন বিষয় নয়।

এই উদাহরণটি হল engineering metaphors এর বিপদের একটি দরকারী নমুনা: মাঝেমাঝে আমরা কম্পিউটারে কিছু বিধিনিষেধ আরোপ করি যা অপ্রয়োজনীয়, অথবা ক্ষমতা প্রত্যাশা করি যা এটিতে নেই, কারন আমরা অচিন্তনীয় ভাবেই এটির ব্রেকিং পয়েন্টের পূর্বে একটি metaphor এক্সটেন্ড করি।

১৪.৬ মার্জসর্ট

সেকশন ১৪.৩ এ, আমরা দেখেছি একটি সাধারণ সর্টিং অ্যালগরিদম যা খুব বেশী কার্যকর নয়। *n* আইটেমসূহ সর্টিং করার জন্য, এটিকে *n* বার অ্যারে ট্র্যাভেরসাল করতে হয়, এবং প্রতিবার ট্র্যাভেরসাল করতে এটি *n* এর সমানুপাতিক একটি পরিমান সময় নেয়। সর্বমোট সময়, সেই কারনে, n^2 এর সমানুপাতিক হয়।

এই সেকশনে আমি একটি আরো বেশী দরকারী mergesort নামক একটি অ্যালগরিদম স্কেচ করেছি। *n* বার সর্ট করার জন্য mergesort, $n \log n$ এর সমানুপাতিক সময় নেয়। সেটিকে চমৎকার বলে মনে নাও হতে পারে, কিন্তু যেহেতু *n* আকারে বড় হয়, তাই n^2 এবং n log n এর পার্থক্য বিশাল হতে পারে। *n* এর কিছু মান চেষ্টা করে দেখ।

mergesort এর অন্তরালে কিছু প্রাথমিক ধারণা হল: যদি তোমার দুটি সাবডেক থাকে, যেগুলোর প্রতিটি সর্টেড করা আছে, সেগুলো একটি সিঙ্গেল সর্টেড ডেক এ মার্জ করা সহজ। একটি কার্ডের ডেক নিয়ে চেষ্টা করে দেখ:

1. প্রতি ১০টি কার্ডের সাথে দুটি সাবডেক থেকে সর্ট করা হয় যখন তারা উপরের সর্বনিম্ন কার্ডগুলোর মুখোমুখি

হয়। উভয় ডেকের অবস্থান আপনার মুখোমুখি করুন।

2. প্রতিটি ডেক থেকে উপরের কার্ডটি তুলনা কর এবং নিচেরটি পছন্দ কর। এটির উপর ফ্লিপ কর এবং মার্জকৃত ডেকে এটি সংযুক্ত কর।
3. দ্বিতীয় স্টেপ পুনরাবৃত্তি কর যতক্ষণ পর্যন্ত ডেক খালি না হয়। এরপর অবশিষ্ট ডেকগুলো মার্জকৃত ডেকে সংযুক্ত কর।

ফলাফলটি হতে পারে একটি একক সর্টেড ডেক। pseudocode এ এটি যেমন দেখায়:

```
public static Deck merge(Deck d1, Deck d2) {
        // create a new deck big enough for all the cards
        Deck result = new Deck(d1.cards.length + d2.cards.length);

        // use the index i to keep track of where we are in
        // the first deck, and the index j for the second deck
        int i = 0;
        int j = 0;

        // the index k traverses the result deck
        for (int k = 0; k < result.cards.length; k++) {

                // if d1 is empty, d2 wins; if d2 is empty, d1 wins;
                // otherwise, compare the two cards

                // add the winner to the new deck
        }
        return result;
}
```

মার্জ পরীক্ষা করার উত্তম উপায় হল একটি ডেক তৈরি এবং শাফল করা, দুটি হ্যান্ড (ছোট) গঠনের জন্য সাবডেক ব্যবহার কর, এবং এরপর পূর্বের অধ্যায় হতে দুটি halves সর্ট করার জন্য সর্ট রুটিন ব্যবহার কর। এরপর তুমি এটি কাজ করে কিনা তা দেখতে মার্জ করার জন্য দুটি halves পাস করতে পারবে।

যদি তুমি দেখ যে এটি কাজ করছে, তাহলে margeSort এর সাধারণ একটি প্রয়োগ চেষ্টা কর:

```
public static Deck mergeSort(Deck deck) {
        // find the midpoint of the deck
        // divide the deck into two subdecks
        // sort the subdecks using sortDeck
        // merge the two halves and return the result
}
```

এরপর যদি তুমি দেখ যে এটি আসলেই কাজ করছে, তাহলে আসল মজা শুরু হল! mergesort এর যাদুকরী বিষয় হল যে এটি পুনরাবৃত্ত হয়। পয়েন্টে হল তুমি কোথায় সাবডেক সর্ট করতে চাও, কেন তুমি পুরাতন কে ডাকবে যা

সর্টের ধীরগতির সংস্করণ? তুমি যেহেতু লেখার প্রক্রিয়ায় আছ কেন তুমি spiffy new mergeSort ডাকছ না?

এটি শুধুমাত্র একটি ভাল ধারণা নয়, আমি অঙ্গিকার করছি ভাল কৃতিত্বের সুবিধা অর্জনের জন্য এটি প্রয়োজন। কিন্তু এটিকে কার্যকর করার জন্য তোমার একটি বেইস কেস থাকতে হবে; অন্যথায় এটি সবসময়ের জন্য পুনরাবৃত্ত হবে। একটি সাধারণ বেস কেস হল 0 অথবা 1 কার্ডের সহিত একটি সাবডেক। যদি mergesort একটি সাধারণ সাবডেক গ্রহন করে, এটি ইতোমধ্যে সর্টেড হলেও অপরিবর্তনীয় ভাবে ফেরত দিতে পারবে।

mergesort এর রিকারসিভ সংস্করণটি এরকম দেখতে:

```
public static Deck mergeSort(Deck deck) {
        // if the deck is 0 or 1 cards, return it
        // find the midpoint of the deck
        // divide the deck into two subdecks
        // sort the subdecks using mergesort
        // merge the two halves and return the result
}
```

যথারীতি, দুটি পথ আছে রিকারসিভ প্রোগ্রাম চিন্তা করার জন্য: তুমি সমস্ত সঞ্চালন প্রবাহের মাধ্যমে চিন্তা করতে পার আবার তুমি "leap of faith" তৈরি করতে পার (সেকশন ৬.৯ দেখ)। আমি leap of faith তৈরি করতে তোমাকে উৎসাহিত করার জন্য এই উদাহরণটি তৈরি করেছি।

সাবডেক সর্ট করার জন্য যখন তুমি sortDeck ব্যবহার কর, তুমি সঞ্চালন প্রবাহ অনুস্মরণ করতে বাধ্য নও, তাই নয় কি? তুমি শুধু ধরে নিয়েছ এটি কাজ করছে কারন তুমি এটি ইতোমধ্যেই ডিবাগ করেছ। ভাল, সমস্ত কিছুই তুমি করেছ mergeSort পুনরাবৃত্তির জন্য যাতে একটি সর্টিং অ্যালগরিদম আরেকটির সাথে প্রতিস্থাপন হয়। অন্যভাবে প্রোগ্রাম রিড করার অন্য কোন কারন নেই।

প্রকৃতঅর্থে, বেস কেস সঠিকভাবে পাওয়ার জন্য তোমাকে কিছু চিন্তা করতে হবে তাইনা? এবং তুমি এটি নিশ্চিত হও যে শেষ পর্যন্ত তুমি এটি করতে পেরেছ, কিন্তু অন্যথায়, রিকারসিভ সংস্করণ লেখা যেতে পারে তাতে কোন সমস্যা নেই। তোমার ভাগ্য সুপ্রসন্ন হোক!

১৪.৭ ক্লাস ভ্যারিয়েবল

এ পর্যন্ত আমরা লোকাল ভ্যারিয়েবল দেখেছি, যা একটি মেথডের ভেতর ডিক্লেয়ার করা হত, এবং ইনসট্যান্স ভ্যারিয়েবল ডিক্লেয়ার করা হয় সাধারণত মেথড ডেফিনেশনের পূর্বে একটি ক্লাস ডেফিনেশনে।

লোকাল ভ্যারিয়েবল তৈরি করা হয় যখন একটি মেথড শেষের সময়ে তা আহ্বান এবং ধ্বংস করা হয়। তুমি একটি অবজেক্ট তৈরির সময় ইনসট্যান্স ভ্যারিয়েবল তৈরি হয় এবং ধ্বংস হয় যখন অবজেক্টটি গার্বেজ হিসেবে সংগৃহীত হয়।

এখন সময় হল ক্লাস ভ্যারিয়েবল এর সম্পর্কে শেখার। ইনসট্যান্স ভ্যারিয়েবল এর মত, ক্লাস ভ্যারিয়েবল একটি মেথড ডেফিনেশনের পূর্বে একটি ক্লাস ডেফিনেশনের মধ্যে ডিফাইন করা হয়, কিন্তু তাদেরকে চিহ্নিত করা হয় কিওয়ার্ড static দ্বারা। তাদেরকে তৈরি করা হয় যখন প্রোগ্রাম শুরু করা হয় এবং টিকে থাকে প্রোগ্রাম শেষ হওয়া পর্যন্ত।

তুমি একটি ক্লাস ডেফিনেশনে যেকোন জায়গা থেকে একটি ক্লাস ভ্যারিয়েবল উল্লেখ করতে পার। ক্লাস ভ্যারিয়েবল মাঝেমাঝে কনসট্যান্ট ভ্যালু সংরক্ষণের জন্য ব্যবহৃত হয় যা কিছুকিছু জায়গায় প্রয়োজন হয়।

উদাহরণ হিসাবে, এখানে Card এর একটি সংস্করণ আছে যেখানে suits এবং ranks হল ক্লাস ভ্যারিয়েবল:

```
class Card {
        int suit, rank;
        static String[] suits = { "Clubs", "Diamonds", "Hearts", "Spades" };
        static String[] ranks = { "narf", "Ace", "2", "3", "4", "5", "6","7", "8", "9", "10",
                                        "Jack", "Queen", "King" };
        public static void printCard(Card c) {
                System.out.println(ranks[c.rank] + " of " + suits[c.suit]);
        }
}
```

যেহেতু তারা লোকাল ভ্যারিয়েবল তাই আমরা printCard এর ভিতরে suits এবং ranks উল্লেখ করতে পারি।

১৪.৮ শব্দকোষ

pseudocode: ইংরেজি এবং জাভা একত্রিত করে খসরা লেখার মাধ্যমে প্রোগ্রাম ডিজাইন করার একটি উপায়।

helper method: প্রায়শই একটি ছোট মেথড নিজের জন্য তেমন কিছুই করে না, কিন্তু অন্যকে সহায়তা করে, যা অতি সহায়ক একটি মেথড।

class variable: static হিসেবে ক্লাসের ভেতর একটি ভ্যারিয়েবল ডিক্লেয়ার করা; বাস্তবে এই ভ্যারিয়েবলের সর্বদা একটি অনুলিপি থাকে।

১৪.৯ অনুশীলনী

অনুশীলনী ১৪.১. এই অধ্যায়ে এই অনুশীলনীর উদ্দেশ্য হল অ্যালগরিদম শাফলিং এবং সর্টিং প্রয়োগ করা।

১। http://thinkapjava.com/code/Card2.java থেকে এই অধ্যায়ের কোডটি ডাউনলোড কর এবং এনভায়রনমেন্ট পরিবেশে ইমপোর্ট কর। তুমি যে মেথডটি লিখবে সেটির প্রোগ্রাম কম্পাইলের জন্য আমি আউটলাইন প্রদান করেছি। কিন্তু যখন এটি রান হবে তখন এটি বার্তা প্রিন্ট করবে যে খালি মেথডটি কাজ করছে না। যখন তুমি তাদের সঠিকভাবে পূরণ করবে তখন বার্তা চলে যাবে।

২। যদি তুমি অনুশীলনী ১২.৩ করে থাক, তুমি তাহলে ইতোমধ্যেই randomInt লিখে ফেলেছ। অন্যথায় এখনই এটি লেখ এবং এটি পরীক্ষার জন্য কোড সংযোগ কর।

৩। swapCards নামক একটি মেথড লেখ যা একটি ডেক (কার্ডের অ্যারে) এবং দুটি সূচক নেয়, এবং যা ঐ দুটি অবস্থানে কার্ডসমূহ সুইচ করে।
ইঙ্গিত: এটি রেফারেন্সসমূহ সুইচ করতে পারে, কিন্তু অবজেক্টের বিষয়বস্তুসমূহ সুইচ করবে না। এটি অনেক দ্রুত

কাজ করে; একই সাথে, এটি সঠিকভাবে কেস নিয়ন্ত্রণ করে যেখানে কার্ডগুলো উপনামে আছে।

৪। shuffleDeck নামক একটি মেথড লেখ যা সেকশন ১৪.২ এর অ্যালগরিদম ব্যবহার করে। তুমি অনুশীলনী ১৪.২ থেকে randomInt মেথড ব্যবহার করতে পার।

৫। indexLiwestCard নামক একটি মেথড লেখ যা ডেকে একটি প্রদত্ত রেঞ্জের মধ্যে সর্বনিম্ন কার্ড অনুসন্ধানের জন্য compareCard মেথড ব্যবহার করে (lowIndex হতে highIndex থেকে, উভয় সহ)।

৬। sortDeck নামক একটি মেথড লেখ যা নিম্ন থেকে উচ্চ কার্ডের একটি ডেক সাজাবে।

৭। সেকশন ১৪.৬ এর pseudocode ব্যবহার করে merge নামক একটি মেথড লেখ। একটি mergeSort এর অংশ হিসাবে এটি ব্যবহার করার চেষ্টার পূর্বে এটি পরীক্ষা করে নিশ্চিত হও।

৮। mergeSort এর সাধারণ সংস্করণ লেখ যা ডেকটিকে অর্ধেক বিভাজন করে, দুটি অর্ধেককে সর্ট করার জন্য sortDeck ব্যবহার করে, এবং একটি new সম্পূর্ণ সর্টেড ডেক তৈরির জন্য merge ব্যবহার করে।

৯। mergeSort এর সম্পূর্ণ রিকারসিভ সংস্করণ লেখ। স্মরণে রাখ যে sortDeck একটি পরিবর্তক এবং mergeSort একটি ফাংশন, যার অর্থ হল তাদের ভিন্ন ভাবে আহ্বান করা হয়:

```
sortDeck(deck); // modifies existing deck
deck = mergeSort(deck); // replaces old deck with new
```

অধ্যায় ১৫

অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং (Object-oriented programming)

১৫.১ প্রোগ্রামিং ল্যাঙ্গুয়েজ এবং স্টাইল

অনেকগুলো প্রোগ্রামিং ল্যাঙ্গুয়েজ এবং এর অনেকগুলো প্রোগ্রামিং শৈলি আছে (অনেক সময় তাদের বলা হয় প্যারাডিজম)। আমরা যে প্রোগ্রামগুলো লিখি তা অনেকটা পদ্ধতিগত নিয়মে লেখা হয়, কারণ এখানে জোর দেওয়া হয় নির্দিষ্ট কিছু গাণিতিক পদ্ধতির উপর।

অধিকাংশ জাভা প্রোগ্রামিং অবজেক্ট ওরিয়েন্টেড, যার অর্থ জাভা অবজেক্ট এবং তাদের মিথস্ক্রিয়ার উপর বেশি ফোকাস করে। এখানে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং এর কিছু বৈশিষ্ট্য দেওয়া হল:

- অবজেক্ট প্রায়শই বাস্তব দুনিয়ার অস্তিত্বকে উপস্থাপন করে। পূর্ববর্তী অধ্যায়ে, আমরা যে Deck ক্লাস তৈরি করেছি তা ছিল অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিংয়ের একটি ধাপ।

- ক্লাস মেথড (স্ট্রিং এ যে মেথডগুলো যুক্ত করেছিলে সেই রকম) ব্যাতিত অধিকাংশ মেথডই অবজেক্ট মেথড (যেমন ম্যাথ মেথড)। আমরা এ পর্যন্ত যে মেথড রাইট করেছি তার বেশির ভাগই ক্লাস মেথড। এই অধ্যায়ে আমরা কিছু অবজেক্ট মেথড রাইট করব।

- যে উপায়ে অবজেক্টগুলো সম্পৃক্ত তাদের সীমাবদ্ধতার কারণে তারা একটি থেকে আরেকটি বিচ্ছিন্ন হয়, বিশেষ করে, মেথড যুক্ত করা ছাড়া ইনসট্যান্স ভ্যারিয়েবল থেকে এদেরকে প্রতিরোধ করতে।
-
- ক্লাসগুলো সংগঠিত হয় একটি ফ্যামিলি ট্রি এ যেখানে নতুন ক্লাস বিদ্যমান ক্লাসে প্রসারিত হয়, নতুন মেথড যুক্ত করে এবং অন্যকে প্রতিস্থাপিত করে।

এই অধ্যায়ে, আমি পূর্ববর্তী অধ্যায়ের অবজেক্ট ওরিয়েন্টেড পদ্ধতি থেকে কার্ড প্রোগ্রাম অনুবাদ করেছি। তুমি এই অধ্যায়ের http://thinkapjava.com/code/Card3.java. এই লিংক থেকে কোডগুলো ডাউনলোড করে নিতে পার।

১৫.২ অবজেক্ট মেথড এবং ক্লাস মেথড (Object methods and class methods)

জাভাতে দুই ধরনের মেথড বিদ্যমান আছে, এদেরকে ক্লাস মেথড এবং অবজেক্ট মেথড বলা হয়। ক্লাস মেথড চেনা পরিচিত হয় প্রথম লাইনের স্ট্যাটিক (static) কীওয়ার্ড দ্বারা। কোন মেথডের যদি স্ট্যাটিক কীওয়ার্ড না থাকে তাহলে তাকে অবজেক্ট মেথড বলা হয়।

যদিও আমরা অবজেক্ট মেথড রাইট করিনি, আমরা কিছু যুক্ত করেছি। যখন তুমি একটি অবজেক্টে “on” মেথড যুক্ত করবে, এটি তখন একটি অবজেক্ট মেথড হবে। উদাহরণস্বরূপ, আমরা স্টিং অবজেক্টে যে *charAt* এবং অন্যান্য মেথড যুক্ত করেছি তার সবই অবজেক্ট মেথড।

একটি ক্লাস মেথড হিসাবে যা কিছু লেখা যায় তা মেথড হিসাবেও লেখা যায় এবং তার উল্টোটাও হতে পারে। কিন্তু কখনও কখনও এটার একটি বা অন্যটি ব্যবহার করা অনেক বেশি প্রাকৃতিক।

উদাহরণস্বরূপ, এখানে printCard একটি ক্লাস মেথড হিসাবে:

```
public static void printCard(Card c) {
        System.out.println(ranks[c.rank] + " of " + suits[c.suit]);
}
```

এখানে এটিকে অবজেক্ট মেথড হিসেবে পুনরায় রাইট করা হল:

```
public void print() {
        System.out.println(ranks[rank] + " of " + suits[suit]);
}
```

এখানে পরিবর্তনগুলো হল:

- আমি স্ট্যাটিক পরিবর্তন করেছিলাম।
- আমি আরও বেশি রীতিসিদ্ধ করতে মেথডের নাম পরিবর্তন করেছিলাম।
- আমি প্যারামিটার পরিবর্তন করেছিলাম।
- একটি অবজেক্ট মেথডের ভেতর তুমি একটি ইনসট্যান্স ভ্যারিয়েবল ব্যবহার করতে পার যদি সেগুলো

লোকাল ভ্যারিয়েবল হয় । তাই আমি c.rank কে পরিবর্তন করেছি rank এ এবং অনুরূপভাবে suit এর জন্য।

এই মেথডগুলো কিভাবে যুক্ত হল তা দেখানো হল;

```
Card card = new Card(1, 1);
card.print();
```

যখন তুমি একটি অবজেক্টে একটি মেথড যুক্ত কর, তখন অবজেক্ট হয়ে পড়ে current অবজেক্ট, এটা this হিসাবেও পরিচিত। print এর ভেতর, this কীওয়ার্ড কার্ড যে মেথডে যুক্ত হয়েছে তা চিহ্নিত করে ।

১৫.৩ toString মেথড

প্রতিটি অবজেক্ট টাইপের একটি মেথড আছে যার নাম toString যা একটি স্টিং রিটার্ন করে এবং এটি একটি অবজেক্টের প্রতিনিধিত্ব করে। যখন তুমি print অথব println ব্যবহার করে একটি অবজেক্ট প্রিন্ট করবে জাভা তখন অবজেক্টের toString মেথড ব্যবহার করবে।

tostring এর পূর্বনির্ধারিত ভার্সন একটি স্ট্রিং রিটার্ন করে যেটা অবজেক্টের টাইপ এবং ইউনিক আইডেনটি ফায়ার (১১.৬ অংশ দেখ) ধারণ করে। যখন তুমি একটি নতুন অবজেক্ট টাইপ নির্ধারণ করবে, তখন তুমি পূর্বনির্ধারিত আচরণ override করতে পারবে। তুমি যে আচরণ চাও তা প্রদান করার মাধ্যমে তুমি এটা করতে পারবে।

উদাহরণস্বরূপ, এখানে card এর জন্য একটি toString মেথড:

```
public String toString() {
        return ranks[rank] + " of " + suits[suit];
}
```

সাধারণভাবেই রিটার্ন এর ধরন হল String, এবং এটা কোন প্যারামিটার গ্রহণ করেনা। তুমি সাধারণ উপায়েই toString ব্যবহার করতে পার:

```
Card card = new Card(1, 1);
String s = card.toString();
```

অথবা তুমি পরোক্ষভাবেও println এর মাধ্যমে এটি যুক্ত করতে পার:

```
System.out.println(card);
```

১৫.৪ ইকুয়াল মেথড (The equals method)

সেকশন ১৩.৪ এ আমরা সমতা সম্পর্কে দুইটি ধারণার কথা বলেছি: পরিচয়, যার মানে হল যে, দুইটি ভ্যারিয়েবল একই অবজেক্টকে রেফার করে এবং সমানতা, যার অর্থ তাদের একই মান আছে।

== অপারেটর পরিচয় পরীক্ষা করে, কিন্তু সমানতা পরিমাপের কোন অপারেটর নাই, কারণ “সাম্য” (equivalence) অবজেক্ট টাইপের উপর নির্ভর করে। পরিবর্তে, অবজেক্ট ইকুয়াল নামে একটি মেথড প্রদান করে

যা সাম্যকে নির্দেশ করে।

জাভা ক্লাস equals মেথড প্রদান করে যেটা সঠিক কাজটি করতে পারে। কিন্তু ব্যবহারকারী কর্তৃক প্রদত্ত ধরনের জন্য পূর্বনির্ধারিত আচরণ পরিচয়ের মতোই হয়।, যেটা আসলে তুমি চাওনা।

Card এর জন্য আমাদের আগেই একটি মেথড আছে যেটা সাম্যতা পরীক্ষা করে:

```
public static boolean sameCard(Card c1, Card c2) {
        return (c1.suit == c2.suit && c1.rank == c2.rank);
}
```

তাই সবকিছু যা আমরা করলাম তা একটি অবজেক্ট মেথডের রিরাইট করা:

```
public boolean equals(Card c2) {
        return (suit == c2.suit && rank == c2.rank);
}
```

পুনরায়, আমি static এবং প্রথম প্যারামিটার, c1 বাদ দিয়েছিলাম। এখানে এটা যুক্ত হয়েছে তা দেখাচ্ছে :

```
Card card = new Card(1, 1);
Card card2 = new Card(1, 1);
System.out.println(card.equals(card2));
```

equals এর ভেতর, card হল current অবজেক্ট এবং card2 হল c2 এর প্যারামিটার। একই ধরনের দুইটি অবজেক্টকে অপারেট করতে যে মেথড, অনেক সময় আমি বিশদ ভাবে ব্যবহার করি এবং এই ভাবে প্যারামিটার কল করি:

```
public boolean equals(Card that) {
        return (this.suit == that.suit && this.rank == that.rank);
}
```

আমি মনে করি এটা পাঠযোগ্যতার উন্নয়ন করবে।

১৫.৫ বৈষম্য এবং ত্রুটি (Oddities and errors)

যদি তোমার একই ক্লাসে অবজেক্ট মেথড এবং ক্লাস মেথড থাকে, তবে এটা সহজেই তোমাকে বিভ্রান্তিতে ফেলবে। ক্লাস ডেফিনেশনগুলোকে সাজানোর একটি সাধারণ উপায় হল সকল কনসট্রাক্টরকে শুরুতে রাখা, সব অবজেক্ট মেথডকে অনুসরণ করা এবং এরপর সব ক্লাস মেথডকে অনুসরণ করা।

তোমার একই নামের একটি অবজেক্ট মেথড এবং একটি ক্লাস মেথড থাকতে পারবে না যতক্ষণ তোমার প্যারামিটারের নম্বর ও ধরন এক না হয়। ওভারলোডিংয়ের অন্যান্য ধরনে, তুমি যে আর্গুমেন্টটা দিয়েছ তা কোন

ভার্সনটা কে যুক্ত করবে সেটা জাভা দেখে ঠিক করে নেয়।

আমরা এখন জানি static কীওয়ার্ড কি, তুমি সম্ভবত এই main কে একটি ক্লাস মেথড হিসাবে চিহ্নিত করতে পেরেছ, যেটা বোঝায়ে যে, যখন তুমি এটা চাও তখন কোন "current object" থাকে না। যেহেতু ক্লাস মেথডে কোন current অবজেক্ট থাকে না, তাই this কীওয়ার্ড ব্যবহার করাটা একটা ত্রুটি। যদি তুমি চেষ্টা কর, তুমি এই রকম একটি এরর মেসেজ পাবে: "Undefined variable: this."

এছাড়াও, তুমি কোন ডট নোটেশন এবং অবজেক্ট নাম ছাড়া ইনসট্যান্স ভ্যারিয়েবল উল্লেখ করতে পারবে না। যদি তুমি করতে চেষ্টা কর, তবে তুমি এই রকম একটি মেসেজ পাবে: "non-static variable... cannot be referenced from a static context." এখানে "non-static variable" দ্বারা "instance variable." বোঝানো হয়েছে।

১৫.৬ ইনহেরিটেন্স (Inheritance)

ইনহেরিটেন্স হল ল্যাঙ্গুয়েজের বৈশিষ্ট্য যা প্রায়ই অবজেক্ট-ওরিয়েন্টেড এর সাথে যুক্ত হয়। একটি নতুন ক্লাস যেটা একটি বিদ্যমান ক্লাসের পরিমার্জিত সংস্করণ তা সংজ্ঞায়িত করার ক্ষমত হল ইনহেরিটেন্স। মেটাফোর সম্প্রসারণে, মাঝে মাঝে বিদ্যমান ক্লাসকে বলা হয় প্যারেন্ট ক্লাস এবং নতুন ক্লাসকে বলা হয় চাইল্ড।

এই বৈশিষ্ট্যের প্রাথমিক সুবিধা হল যে, তুমি প্যারেন্ট কে মডিফাই না করেই মেথড এবং ইনসট্যান্স ভ্যারিয়েবল যুক্ত করতে পারবে। এটি জাভা ক্লাসের জন্য বিশেষভাবে দরকারি, কেননা তুমি চাইলেও এদের পরিবর্তন করতে পারবে না।

তুমি যদি GridWorld এর অনুশীলনীগুলো করে থাক (অধ্যায় ৫ থেকে ১০) তবে তুমি ইনহেরিটেন্স এর উদাহরণগুলো দেখে থাকবে:

```
public class BoxBug extends Bug {
        private int steps;
        private int sideLength;

        public BoxBug(int length) {
                steps = 0;
                sideLength = length;
        }
}
```

BoxBug extends Bug এর অর্থ হল BoxBug একটি নতুন ধরনের Bug যেটা মেথড এবং ইনসট্যান্স ভ্যারিয়েবলের Bug.In কে সংযোজন করতে সক্ষম :

- চাইল্ড ক্লাসের অতিরিক্ত ইনসট্যান্স ভ্যারিয়েবল থাকতে পারে; এই উদাহরণে, BoxBugs এর steps এবং sideLength থাকে।
- চাইল্ডের এডিশনাল মেথড থাকতে পারে, এই উদাহরণে, BoxBugs এর একটি অতিরিক্ত কনসট্রাকটর আছে যেটা একটি ইনটিজার প্যারামিটার গ্রহণ করে।
- চাইল্ড প্যারেন্টের কাছ থেকে কোন মেথড override করতে পারে; এই উদাহরণে, চাইল্ড act প্রদান করে

(এখানে দেখানো হয়নি), যেটা প্যারেন্ট থেকে act মেথডকে override করে।

যদি তুমি Appendix A এর গ্রাফিক্সের উদাহরণগুলো কর, তুমি তাহলে অন্য একটি উদাহরণ দেখবে:

```
public class MyCanvas extends Canvas {

        public void paint(Graphics g) {
                g.fillOval(100, 100, 200, 200);
        }
}
```

MyCanvas একটি নতুন ধরনের canvas যার মধ্যে কোন নতুন মেথড অথবা ইনসট্যান্স ভ্যারিয়েবল নাই কিন্তু এটি paint কে ওভাররাইড করে।

তুমি যদি আগের উদাহরণগুলোতে এটার অনুশীলনী না করে থাক তবে এটি করার জন্য এখন উত্তম সময়!

১৫.৭ ক্লাসের অনুক্রম (The class hierarchy)

জাভায় সব ক্লাসগুলো অন্যান্য ক্লাসে প্রসারিত হয়। অধিকাংশ মৌলিক ক্লাসকে অবজেক্ট বলা হয়। এটা কোন ইনসট্যান্স ভ্যারিয়েবল ধারণ করে না, কিন্তু এটি অন্যদের মাঝে equals এবং toString মেথড প্রদান করে।

অনেকগুলো ক্লাস সহ, আমরা যেসব ক্লাস রাইট করেছি এবং এর সাথে অনেক জাভা ক্লাস যেমন- java.awt.Rectangle অবজেক্টকে প্রসারিত করে। যে ক্লাস স্পষ্টভাবে একটি প্যারেন্টর নাম দিতে পারে না সেই সকল ক্লাস পূর্বনির্ধারিতভাবে অবজেক্ট থেকে ইনহেরিট করে থাকে।

যদিও কিছু ইনহেরিট্যান্স চেইন অনেক দীর্ঘ। উদাহরণস্বরূপ, javax.swing.JFrame যেটা প্রসারিত করে java.awt.Frame কে, যেটা প্রসারিত করে Window কে, যেটা প্রসারিত করে Container কে, যেটা প্রসারিত করে Component কে, যেটা প্রসারিত করে Object কে। তাই বলা যায় চেইন যত বড়ই হোক না কেন, এদের মূল হল অবজেক্ট।

ক্লাসের “ family tree” কে বলা হয় ক্লাসের অনুক্রম। অবজেক্ট সাধারণত সবার উপরে প্রদর্শিত হয়, এর নিচে সবগুলো “child” থাকে। তুমি যদি উদাহরণের জন্য JFrame এর ডকুমেন্টেশন দেখ, তুমি অনুক্রমের একটি অংশ দেখতে পারবে যেটা JFrame এর বংশতালিকা তৈরি করে।

১৫.৮ অবজেক্ট-ওরিয়েন্টেড ডিজাইন (Object-oriented design)

ইনহেরিট্যান্স একটি শক্তিশালী বৈশিষ্ট্য। যেসব প্রোগ্রামগুলো ইনহেরিট্যান্স ছাড়া জটিল তা এটির সাহায্যে সংক্ষিপ্ত ও সহজ করে রাইট করা যায়। এছাড়াও ইনহেরিট্যান্স কোডকে পুনরায় ব্যবহারের সুযোগ দেয়, যাতে তুমি বিদ্যমান ক্লাসের চরিত্র কোন পরিবর্তন করা ছাড়াই কাস্টমাইজড করতে পার।

অন্য দিকে, ইনহেরিট্যান্স প্রোগ্রাম রিড করতে কঠিন করে তুলে। যখন তুমি একটি মেথড আহ্বান করতে দেখ, তখন এটা খুঁজে বের করা কষ্টকর হবে যে কোন মেথড যুক্ত করা হয়েছে।

এছাড়াও, যে কাজগুলো ইনহেরিট্যান্স দিয়ে করা যায় তা এটা দিয়েও করা যায়। একটি সাধারণ বিকল্প হল

কম্পোজিশন, যেখানে নতুন অবজেক্ট বিদ্যমান অবজেক্ট দিয়ে কম্পোজড করা হয়, ইনহেরিট্যান্স ছাড়াই নতুন ক্যাপাবিলিটি যোগ করা হয়।

অবজেক্ট ডিজাইন করা এবং এদের মাঝে সম্পর্কই হল অবজেক্ট-ওরিয়েন্টেড ডিজাইন, যা এই বইয়ে দেওয়া যায়নি। কিন্তু তুমি যদি আগ্রহী হও, তবে আমি তোমাকে আরেকটি বই পড়তে বলব- *Head First Design Patterns,* যেটা প্রকাশ করেছিল O'Reilly Media।

১৫.৯ শব্দকোষ (Glossary)

অবজেক্ট মেথড (object method): একটি মেথড যা একটি অবজেক্টে যুক্ত হয়, এবং এটি অবজেক্টকে অপারেট করতে পারে। অবজেক্ট মেথডের static কীওয়ার্ড থাকে না।

ক্লাস মেথড (class method): static কীওয়ার্ড সহকারে একটি মেথড। ক্লাস মেথড অবজেক্টে যুক্ত হয় না এবং তাদের current অবজেক্ট থাকে না।

কারেন্ট অবজেক্ট (current object): অবজেক্টের উপর যে অবজেক্ট যুক্ত হয়। মেথডের ভেতর, কারেন্ট অবজেক্ট দ্বারা this রেফার্ড করা হয়।

ইমপ্লিসিট (implicit): যে কোন কিছু যা অনুক্ত অথবা উহ্য। একটি অবজেক্ট মেথডের মধ্যে, তুমি ইনসট্যান্স ভ্যারিয়েবল ইমপ্লিসিটি রেফার করতে পার (যেমন, অবজেক্ট ছাড়া নামকরণ)।

এক্সপ্লিসিট (explicit): এমন কিছু যা সম্পূর্ণ উচ্চারিত হয়। একটি ক্লাস মেথডের মধ্যে, ইনসট্যান্স ভ্যারিয়েবলের সব রেফারেন্স হবে এক্সপ্লিসিট।

অনুশীলনী ১৫.১০

অনুশীলনী ১৫.১: ডাউনলোড কর

http://thinkapjava.com/code/CardSoln2.java এবং
http://thinkapjava.com/code/CardSoln3.java
CardSoln2.java পূর্ববর্তী অধ্যায়ের অনুশীলনীর উদাহরণ ধারণ করে। এটা শুধু মাত্র ক্লাস মেথড ব্যবহার করে (কনসট্রাকটর ছাড়া)।

CardSoln3.java একই প্রোগ্রাম গ্রহণ করে, কিন্তু অধিকাংশ মেথড হল অবজেক্ট মেথড। আমি merge কে পরিবর্তন করাই ছেড়েছি কারণ আমি চিন্তা করেছি এটা ক্লাস মেথড হিসাবে অনেক বেশি পড়ার যোগ্য।
merge কে একটি অবজেক্ট মেথডে রূপান্তর এবং mergeSort এ পরিবর্তন করা। তুমি merge এর কোন ভার্সন কে পছন্দ করবে?

অনুশীলনী ১৫.২: নিচের ক্লাস মেথডকে অবজেক্ট মেথডে রূপান্তর কর।

```
public static double abs(Complex c) {
        return Math.sqrt(c.real * c.real + c.imag * c.imag);
}
```

অনুশীলনী ১৫.৩: নিচের ক্লাস মেথডকে অবজেক্ট মেথডে রূপান্তর কর।

```
public boolean equals(Complex b) {
        return(real == b.real && imag == b.imag);
}
```

অনুশীলনী ১৫.৪: এই অনুশীলনীটি পূর্ববর্তী অধ্যায় ১১.৩ এর ধারাবাহিক অনুশীলনী। এর উদ্দেশ্য অবজেক্ট মেথডের সিনট্যাক্সের অনুশীলনী করা এবং এর সাথে সংশ্লিষ্ট এরর মেসেজের সাথে পরিচিত হওয়া।

1. অবজেক্ট মেথডরে ক্লাস মেথড থেকে Rational মেথডে রূপান্তর কর এবং main এর প্রয়োজনীয় পরিবর্তন কর।
2. কিছু ভুল কর। ক্লাস মেথডকে এমন ভাবে যুক্ত কর যেন তা অবজেক্ট মেথড ছিল এবং এর বিপরীত টাও কর। কোনটি লিগাল এবং কোনটি নয় এবং এরর মেসেজ যেটি তুমি বিশৃঙ্খল অবস্থায় পেয়েছ তার সম্পর্কে একটি ধারণা তৈরি কর।
3. ক্লাস এবং অবজেক্ট মেথডের সুবিধা এবং অসুবিধাগুলো চিন্তা কর। কোনটি সবচেয়ে বেশি সংক্ষিপ্ত (সাধারণত)? কোনটি গণনার বর্ণনার কাজে বেশি প্রাকৃতিক (অথবা, বেশি স্বচ্ছ, প্রতিটি স্টাইল ব্যবহারে কোন ধরনের গণনা পরিষ্কারভাবে ধারণা দিতে পারে)?

অনুশীলনী ১৫.৫: এই অনুশীলনীর উদ্দেশ্য হল একটি প্রোগ্রাম লেখা যেটা র‍্যানডম পোকার হ্যান্ডস (random poker hands) এক্সিকিউট করতে পারে এবং তাদের ক্লাসিফাইড করতে পারে, যাতে আমরা পরিমাপ করতে পারি নানান ধরনের poker hands এর সম্ভাবনার সম্পর্কে। তুমি যদি কখনও poker না খেলে থাক তবে এটা সম্পর্কে এখানে পড়তে পার http://en.wikipedia.org/wiki/List_of_poker_hands

1. http://thinkapjava.com/code/CardSoln3.java এটা দিয়ে শুরু কর এবং এটা নিশ্চিত কর যে, তুমি এটা কম্পাইল করতে পারছ ও রান করতে পারছ।
2. PokerHand ক্লাসের জন্য একটি ডেফিনেশন রাইট কর যেটা Deck কে সম্প্রসারিত করতে পারে।
3. deal নামে একটি Deck মেথড রাইট কর যেটা একটি PokerHand তৈরি করে, কার্ডগুলোকে ডেক থেকে হাতে নাও এবং হাতে রিটার্ন কর।
4. কাজটি করতে main এ shuffle এবং deal ব্যবহার কর এবং প্রতিটির সাথে পাঁচটি কার্ডসহ চারটি PokerHands প্রিন্ট কর। তুমি কি এর চেয়ে ভাল কিছু পাচ্ছ?
5. hasFlush নামে একটি PokerHands মেথড রাইট কর যেটা একটি বুলিয়ান চিহ্নিত করে যতক্ষণ পর্যন্ত হাতে flush ধারণ করা থাকে।
6. hasThreeKind নামে একটি মেথড রাইট কর যেটা চিহ্নিত করে যে হাত তিনটি Kind ধারণ করে আছে।
7. একটি loop রাইট কর যা কয়েক হাজার হ্যান্ডকে উৎপাদন করতে পারে এবং পরীক্ষা করে দেখ যে, তারা একটি flush ধারণ করছে না তিন প্রকারের kind ধারণ করছে। এর যে কোন একটি হ্যান্ডস পাওয়ার সম্ভাবা পরিমাপ কর। তোমার রেজাল্টের সাথে সম্ভাবনার একটি তুলনা এখান থেকে কর http://en.wikipedia.org/wiki/List_of_poker_hands ।
8. একটি মেথড রাইট কর যেটা অন্যান্য পোকার হ্যান্ডসের জন্যও পরীক্ষা করে। কিছু থাকবে যা অন্যগুলো থেকে সহজ হবে। তুমি এর উপকারী দিক খুঁজে পাবে সাধারণ-উদ্দেশ্য রাইটের সাহায্যকারী মেথড হিসাবে যেটা ব্যবহার করা হবে একাধিক পরীক্ষার জন্য।
9. কিছু পোকার গেমস এ, খেলোয়াড়রা প্রত্যেকে সাতটি কার্ড পাবে, এবং তারা সাতটির মধ্যে সেরা পাঁচটি নিয়ে একটি হাত গঠন করতে পারবে। তোমার প্রোগ্রামকে পরিবর্তন কর সাত কার্ড বিশিষ্ট হাতে এবং সম্ভাবনা পুনরায় গণনা করা।

অধ্যায় ১৬
গ্রিডওয়ার্ল্ড: পর্ব ৩ (GridWorld: Part 3)

তুমি যদি অধ্যায় ৫ এবং ১০ এর অনুশীলনী করে না থাক, তাহলে এ অধ্যায় শুরু করার পূর্বে অবশ্যই তা শেষ করে আস। মনে করিয়ে দিচ্ছি, তুমি গ্রিডওয়ার্ল্ড ক্লাস এর ডকুমেন্টেশন পাবে এখানে http://www.greenteapress.com/thinkapjava/javadoc/gridworld/

গ্রিডওয়ার্ল্ড পর্ব ৩ এর স্টুডেন্ট ম্যানুয়াল উপস্থাপন করে এমন সব ক্লাসকে যা গ্রিডওয়ার্ল্ড এবং তাদের মধ্যকার মিথস্ক্রিয়া তৈরি করে । এটি একটি অবজেক্ট ওরিয়েন্টেড ডিজাইনের উদাহরণ এবং OO ডিজাইনের বিভিন্ন দিক আলোচনা করার সুযোগ।

কিন্তু স্টুডেন্ট ম্যানুয়াল পড়ার পূর্বে, কিছু জিনিস রয়েছে যা তোমার জানা প্রয়োজন।

১৬.১ অ্যারে তালিকা (ArrayList)

গ্রিডওয়ার্ল্ড ব্যবহার করে java.util.ArrayList, যা array এর মত আরেকটি অবজেক্ট। এটি একটি সংগ্রহ (collection), যার অর্থ হচ্ছে এটি এমন একটি অবজেক্ট যা অন্য অবজেক্ট ধারণ করে। জাভা ভিন্ন যোগ্যতা সম্পন্ন অন্যান্য সংগ্রহ প্রদান করে থাকে, তবে গ্রিডওয়ার্ল্ড ব্যবহারের জন্য আমাদের প্রয়োজন ArrayLists.

একটি উদাহরণ দেখার জন্য, ডাউনলোড কর http://thinkapjava.com/code/BlueBug.java এবং http://thinkapjava.com/code/BlueBugRunner.java . BlueBug এমন একটি বাগ যা এলোমেলো ভাবে চলাফেরা করে এবং rocks সন্ধান করতে থাকে। যদি এটি rocks পেয়ে যায় তাহলে এটিকে blue করে দেয়।

এটা কীভাবে কাজ করে দেখা যাক। যখন act কে আহ্বান করা হয়, তখন BlueBug এর অবস্থান এবং গ্রিড এর রেফারেন্স পেয়ে যায়:

```
Location loc = getLocation();
Grid<Actor> grid = getGrid();
```

যে দুটো কৌণিক বন্ধনী (<>) দেখতে পাচ্ছ তা হল type parameter যা গ্রিড এর বিষয়বস্তু নির্দিষ্ট করে। অন্যভাবে বলতে গেলে গ্রিড শুধুমাত্র গ্রিড নয়, এটা এমন গ্রিড যা Actor ধারণ করে।

পরবর্তী ধাপ হচ্ছে বর্তমান অবস্থানের প্রতিবেশীকে পাওয়া। গ্রিড এমন একটি মেথড প্রদান করে যা এই কাজটি করে দেয়:

```
ArrayList<Actor> neighbors = grid.getNeighbors(loc);
```

getNeighbors এর রিটার্ন ভ্যালু হচ্ছে একটি Actor এর ArrayList. আকৃতির মেথড ArrayList এর দৈর্ঘ্য রিটার্ন করে, এবং get একটি element নির্বাচন করে। সুতরাং আমরা neighbor কে এইভাবে প্রিন্ট করতে পারি।

```
for (int i = 0; i < neighbors.size(); i++) {
    Actor actor = neighbors.get(i);
```

```
        System.out.println(actor);
    }
```

ArrayList ট্র্যাভার্স করা এতটাই সাধারণ অপারেশন যে এর জন্য একটি সিনট্যাক্স রয়েছে: এটা হল for-each loop. সুতরাং আমরা লিখতে পারি:

```
for (Actor actor : neighbors) {
    System.out.println(actor);
}
```

আমরা জানি যে neighbors হচ্ছে Actors, কিন্তু আমরা এর ধরন জানি না: এটা হতে পারে Bugs, Rocks ইত্যাদি। Rock খোঁজার জন্য আমরা instanceof নামক অপারেটর ব্যবহার করব, যা পরীক্ষা করে দেখে কোন অবজেক্ট ওই ক্লাস এর ইনসট্যান্স কিনা।

```
for (Actor actor : neighbors) {
    if (actor instanceof Rock) {
        actor.setColor(Color.blue);
    }
}
```

এই সবকিছুকে কাজ করাতে হলে, আমাদের কে ব্যবহৃত ক্লাস ইমপোর্ট করতে হবে:

```
import info.gridworld.actor.Actor;
import info.gridworld.actor.Bug;
import info.gridworld.actor.Rock;
import info.gridworld.grid.Grid;
import info.gridworld.grid.Location;

import java.awt.Color;
import java.util.ArrayList;
```

১৬.২ ইন্টারফেস (Interfaces)

গ্রিডওয়ার্ল্ডও জাভা interfaces ব্যবহার করে থাকে, সুতরাং আমি এর কারণ ব্যাখ্যা করতে চাই। "Interface" এর মানে হচ্ছে বিভিন্ন অনুষঙ্গে বিভিন্ন জিনিস, কিন্তু জাভাতে এটার একটি নির্দিষ্ট ল্যাঙ্গুয়েজের বৈশিষ্ট্য উল্লেখ করে: একটি ইন্টারফেস হচ্ছে একটি ক্লাস ডেফিনিশন যেখানে মেথডের কোন body থাকে না।

একটা সাধারণ ক্লাস ডেফিনিশনে, প্রতিটি মেথডের একটি প্রোটোটাইপ এবং একটি body রয়েছে (৮.৫ অনুচ্ছেদ দেখ)। একটি প্রোটোটাইপ কে সবিস্তার বিবরণীও বলা হয় কারণ এটি মেথডের নাম, প্যারামিটার, এবং রিটার্ন টাইপ সুনির্দিষ্ট করে।

জাভার ইন্টারফেসে মেথডের কোন body থাকে না, সুতরাং এটি মেথড বাস্তবায়ন ছাড়াই সুনির্দিষ্ট করে।

উদাহরণস্বরূপ, java.awt.Shape একটি contain, intersects, এবং অন্যান্য মেথডের প্রোটোটাইপ সম্বলিত একটি ইন্টারফেস। java.awt.Rectangle এই মেথডগুলোকে বাস্তবায়ন করে, সুতরাং আমরা বলতে পারি "Rectangle বাস্তবায়ন করে Shape কে"। Rectangle ক্লাস ডেফিনিশন এর প্রথম লাইন হচ্ছে:

```
public class Rectangle extends Rectangle2D implements Shape, Serializable
```
Rectangle মেথড inherit করে Rectangle2D হতে এবং Shape ও Serializable এ মেথড বাস্তবায়ন করে।

গ্রিডওয়ার্ল্ডে Location ক্লাস java.lang.Comparable ইন্টারফেস বাস্তবায়ন করে সেই সাথে compareTo প্রদান করে, যা ১৩.৫ অনুচ্ছেদের compareCards এর অনুরূপ। গ্রিডওয়ার্ল্ড একটি নতুন ইন্টারফেসও ডিফাইন করে, যার নাম Grid, যা কিছু মেথড সুনির্দিষ্ট করে প্রদান করে। এবং এটি দুই রকম বাস্তবায়ন সম্পন্ন করতে পারে, BoundedGrid এবং UnboundedGrid.

স্টুডেন্ট ম্যানুয়াল API নামক একটি সংক্ষিপ্ত শব্দ ব্যবহার করেছে, যার পুরো অর্থ হচ্ছে "application programming interface". অ্যাপ্লিকেশন প্রোগ্রামারের ব্যবহারের জন্য, API হচ্ছে কিছু মেথডের সেট। http://en.wikipedia.org/wiki/Application_programming_interface দেখ।

১৬.৩ পাবলিক এবং প্রাইভেট (public and private)

তোমার মনে আছে কি? অধ্যায় ১ এ আমি বলেছিলাম কেন main মেথডে public কিওয়ার্ড থাকে তা পরে ব্যাখ্যা করব? অবশেষে এটা ব্যাখ্যা করার সময় এসেছে।

public এর মানে হচ্ছে মেথডটি অন্যান্য ক্লাস আহ্বান করতে পারবে। এর উল্টোটি হচ্ছে private, এর মানে হচ্ছে মেথডটি শুধু মাত্র ঐ ক্লাসের মধ্যেই আহ্বান করা যাবে যেখানে এটি ডিফাইন করা হয়েছে।

Instance ভ্যারিয়েবলও public এবং private হতে পারে, এর ফলাফলও একই: একটি private ইনসট্যান্স ভ্যারিয়েবল শুধুমাত্র ঐ ক্লাসেই একসেস করা যাবে যেখানে এটি ডিফাইন করা হয়েছে।

মেথড এবং ইনসট্যান্স ভ্যারিয়েবল private করার প্রধান কারণ হচ্ছে ক্লাসের মধ্যকার মিথস্ক্রিয়া সীমিত করা যাতে জটিলতা দূর হয়।

উদাহরণস্বরূপ, Location ক্লাস তার ইনসট্যান্স ভ্যারিয়েবল private করে রাখে। এর এক্সেসর মেথড রয়েছে getRow এবং getCol, কিন্তু এটি এমন কোন মেথড প্রদান করে না যা ইনসট্যান্স ভ্যারিয়েবল পরিবর্তন করতে সক্ষম। ফলশ্রুতিতে, Location অবজেক্ট অপরিবর্তনীয়, যার অর্থ অ্যালিয়াসিং এর দরুন অপ্রত্যাশিত ব্যবহার এর সমস্যা ছাড়াই এটিকে শেয়ার করা যাবে।

মেথড private করে রাখলে API কে সহজতর করে রাখা সহায়ক হয়। ক্লাস সমূহ প্রায়শই helper মেথড যোগ করে যা অন্যান্য মেথড বাস্তবায়ন করার জন্য ব্যবহৃত হয়, কিন্তু এই মেথডগুলো public করলে API অপ্রয়োজনীয় এবং ত্রুটি-প্রবণ হয়ে যাবে।

Private মেথড এবং ইনসট্যান্স ভ্যারিয়েবল ল্যাঙ্গুয়েজের এমন দুটো বৈশিষ্ট্য যা data encapsulation নিশ্চিত করে থাকে, এর মানে হচ্ছে এক ক্লাসের অবজেক্ট এবং অন্য ক্লাসের অবজেক্ট দুটো আলাদা হয়।

১৬.৪ গেইম অফ লাইফ (Game of Life)

গণিতবিদ জন কনওয়ে "Game of Life" আবিষ্কার করেন, যেটাকে তিনি বলতেন "Zero-player game" কারণ কোন খেলোয়ারকেই কোন ধরনের কৌশল অবলম্বন অথবা নির্ধারণ করতে হত না। প্রাথমিক শর্ত সেটআপ করার পর তুমি দেখবে যে, গেইমটি নিজ থেকে চলছে। শুনতে যতটা না আকর্ষণীয় লাগছে ব্যাপারটি তার থেকেও বেশী মজার; তুমি এই সম্পর্কে পড়তে পারবে এখানে
http://en.wikipedia.org/wiki/Conways_Game_of_Life

এই অনুশীলনের লক্ষ্য হচ্ছে গ্রিডওয়ার্ল্ডে এই গেইম অফ লাইফ বাস্তবায়ন করা। গেইমের বোর্ড হচ্ছে গ্রিড, এবং টুকরো অংশ হচ্ছে রকস।

পরিবর্তন অথবা time steps এর সাথে সাথে গেইম টি অগ্রসর হয়। time step এর শুরুতে, প্রতিটি রক হয় "alive" থাকে অথবা "dead" হয়ে যায়। স্ক্রীনে রক এর রঙ অবস্থা সূচিত করে। প্রতিটি রক এর অবস্থা তার প্রতিবেশীর উপর নির্ভর করে। প্রতিটি রক এর গ্রিড এর কিনারার ছাড়া রয়েছে ৮ টি প্রতিবেশী। নিয়মগুলো হচ্ছে:

- যদি একটি মৃত রক এর তিনটি প্রতিবেশী থাকে, এটি জীবিত হবে! অন্যথায় এটি মৃত অবস্থায় থাকবে।
- যদি একটি জীবিত রক এর ২ অথবা ৩ টি প্রতিবেশী থাকে, এটি টিকে থাকে। অন্যথায় এটি মারা যায়।

এই নিয়মের কিছু ফলাফল: যদি সব রক মারা যায়, কোন রকই আর জীবিত হতে পারে না। যদি তুমি একটি জীবিত রক নিয়ে আরম্ভ কর, এটি মারা যাবে। কিন্তু তোমার যদি ৪ টি রক থাকে, তারা একে অপরকে বাঁচিয়ে রাখবে, সুতরাং এটিই হচ্ছে স্ট্যাবল কনফিগারেশন।

বেশীরভাগ সাধারণ কনফিগারেশন তাড়াতাড়ি নষ্ট হয়ে যায় অথবা স্ট্যাবল হয়। কিন্তু কিছু আরম্ভকারী কনফিগারেশন রয়েছে যা কিছু অসাধারণ জটিলতা প্রদর্শন করে। এর মধ্যে একটি হল r-pentomino: এটি শুরু হয় মাত্র ৫ টি রক দিয়ে, ১১০৩ টাইমস্ট্যাম্পে চলে এবং শেষ হয় ১১৬ টি জীবিত রক নিয়ে স্ট্যাবল অবস্থায়। (http://www.conwaylife.com/wiki/R-pentomino দেখ)

পরবর্তী অংশ হচ্ছে গ্রিডওয়ার্ল্ডে গেইম অফ লাইফ বাস্তবায়ন করার কিছু নির্দেশনা। তুমি আমার করা সমাধানটি ডাউনলোড করে নিতে পার এখান থেকে http://thinkapjava.com/code/LifeRunner.java এবং http://thinkapjava.com/code/LifeRock.java এখান থেকে।

১৬.৫ লাইফরানার (LifeRunner)

BugRunner.java এর একটি অনুলিপি কর এবং নাম দাও LifeRunner.java এবং নিচের প্রোটোটাইপ অনুসারে মেথড যোগ কর:

```
/**
* Makes a Game of Life grid with an r-pentomino.
*/
        public static void makeLifeWorld(int rows, int cols)
/**
* Fills the grid with LifeRocks.
```

```
*/
        public static void makeRocks(ActorWorld world)
```

makeLifeWorld অবশ্যই একটি Actor এবং ActorWorld গ্রিড তৈরি করবে, তারপর makeRocks আহ্বান করবে, যা অবশ্যই LifeRock কে গ্রিড এর প্রতিটি অবস্থানে রাখবে।

১৬.৬ লাইফরক (LifeRock)

Bug.java এর অনুলিপি কর এবং নাম দাও LifeRock.java। LifeRock অবশ্যই Rock কে এক্সটেন্ড করবে। একটি act মেথড যোগ কর যা কিছুই করে না। এই অবস্থায় তুমি কোডটি চালাতে এবং গ্রিড ভর্তি রকস দেখতে সক্ষম হবে।
Rocks এর স্ট্যাটাস ট্র্যাক করে রাখতে, তুমি একটি নতুন ইনসট্যান্স ভেরিয়েবল যোগ করতে পার, অথবা তুমি Rock এর রঙ স্ট্যাটাস ইঙ্গিত দেয়ার জন্য ব্যবহার করতে পার। যেকোন ভাবেই, এই প্রোটোটাইপের সাথে মেথড লিখ:

```
/**
* Returns true if the Rock is alive.
*/
        public boolean isAlive()
/**
* Makes the Rock alive.
*/
        public void setAlive()
/**
* Makes the Rock dead.
*/
        public void setDead()
```

একটি কনস্ট্রাকটর লিখ যা setDead আহ্বান করে এবং নিশ্চিত করে সকল Rocks মারা গিয়েছে।

১৬.৭ এককালীন হালনাগাদ (Simultaneous Updates)

গেম অফ লাইফে, সকল Rocks এককালীন ভাবে হালনাগাদ হয়েছে; অর্থাৎ, প্রতিটি Rock তার স্ট্যাটাস পরিবর্তনের পূর্বে তাদের প্রতিবেশীর স্ট্যাটাস পরীক্ষা করে। অন্যথায় সিস্টেমের ব্যবহার নির্ভর করতো হালনাগাদের ক্রম এর উপর।

এককালীন হালনাগাদ বাস্তবায়ন করতে, আমি পরামর্শ দেব তুমি এমন একটি act মেথড লিখ যার দুটি দশা রয়েছে: প্রথম দশায়, সকল Rocks তাদের প্রতিবেশীর সংখ্যা গুণে তা সংরক্ষণ করে রাখে; দ্বিতীয় দশায়, সকল Rocks তাদের স্ট্যাটাস হালনাগাদ করে।

আমার act মেথডটি দেখত যেমন:

```
    /**
     * Check what phase we're in and calls the appropriate method.
     * Moves to the next phase.
     */
public void act() {
    if (phase == 1) {
        numNeighbors = countLiveNeighbors();
        phase = 2;
    } else {
        updateStatus();
        phase = 1;
    }
}
```

দশা এবং numNeighbors হচ্ছে ইনসট্যান্স ভ্যারিয়েবল। এবং countLiveNeighbors ও updateStatus এর জন্য প্রোটোটাইপ হচ্ছে:

```
    /**
     * Counts the number of live neighbors.
     */
public int countLiveNeighbors()

    /**
     * Updates the status of the Rock (live or dead) based on
     * the number of neighbors.
     */
public void updateStatus()
```

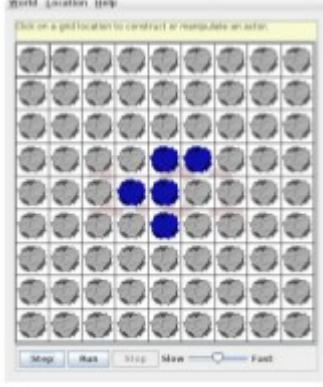
updateStatus এর সাধারণ সংস্করণের দিয়ে শুরু করা যাক যা জীবন্ত Rocks কে মেরে ফেলবে এবং উল্টো টা করবে। এখন প্রোগ্রামটি চালাও এবং নিশ্চিত হও যে Rocks রঙ পরিবর্তন করে। ওয়ার্ল্ডে প্রতিটি ধাপ হচ্ছে গেম অফ লাইফের একটি টাইমস্ট্যাম্প এর সমান।

এখন নিয়ম অনুসারে countLiveNeighbors এবং updateStatus এর বডি পূর্ণ কর এবং দেখ সিস্টেম আশানুরূপ ব্যবহার করছে কিনা।

১৬.৮ প্রাথমিক অবস্থা

প্রাথমিক অবস্থা পরিবর্তনের করার জন্য তুমি গ্রিডওয়ার্ল্ড পপ-আপ মেনু রকস এর স্ট্যাটাস ব্যবহার করতে পার setAlive কে আহ্বান করে। অথবা তুমি প্রক্রিয়া স্বয়ংক্রিয় করার জন্য মেথড লিখতে পার।

LifeRunner এ, একটি মেথড যোগ কর এবং নাম দাও makeRow যা গ্রিড এর মাঝে একটি সারি তে একটি প্রাথমিক কনফিগারেশন তৈরি করবে n সংখ্যক জীবিত Rocks এর সাথে। n এর ভিন্ন ভিন্ন মানের জন্য কি ঘটবে?

MakePentomino নামক একটি মেথড তৈরি কর যা গ্রিড এর মাঝে একটি r-pentomino তৈরি করে। প্রাথমিক অবস্থা দেখতে অনেকটা এরকম:

যদি তুমি এই কনফিগারেশন এক ধাপের বেশী চালাও, এটা গ্রিড এর শেষ প্রান্তে পৌঁছে যায়। গ্রিড এর বাউন্ডারী r-pentomino এর সম্পূর্ণ অভিব্যাক্তি দেখার জন্য সিস্টেমের ব্যবহার পরিবর্তন করে; এর জন্য গ্রিড অবশ্যই বৃহৎ হতে হবে। সঠিক আকৃতি খুঁজে পেতে তুমি পরীক্ষা চালিয়ে দেখতে পার, এটা অবশ্য তোমার কম্পিউটারের শক্তির উপর নির্ভর করে, সুতরাং এই কাজে কিছু সময় ব্যয় হবে।

গেম অফ লাইফ ওয়েব পেজ আরও কিছু প্রাথমিক অবস্থা ব্যাখ্যা করেছে যা আরও চমকপ্রদ (www.conwaylife.com). যেটা তোমার পছন্দ হয় সেটা নির্বাচন কর এবং বাস্তবায়ন কর।

গেম অফ লাইফের কিছু ভ্যারিয়েশনও আছে যা ভিন্ন নিয়মের ভিত্তিতে তৈরি। যে কোন একটা চেষ্টা করে দেখ এবং মজার কিছু খুঁজে বের কর।

১৬.৯ অনুশীলনী

অনুশীলনী ১৬.১: BlueBug.java অনুলিপি করার মাধ্যমে, নতুন ধরনের বাগের জন্য একটি ক্লাস ডেফিনিশন লিখ যা ফুল খুঁজবে এবং খেয়ে ফেলবে। তুমি একটি ফুল খেতে “eat” পারবে removeSelfFromGrid আহ্বান করার মাধ্যমে।

অনুশীলনী ১৬.২: এখন তুমি জান, যেটা তোমার গ্রিড ওয়ার্ল্ড স্টুডেন্ট ম্যানুয়াল এর পর্ব ৩ পড়ার জন্য জানার এবং অনুশীলন করার প্রয়োজন ছিল।

অনুশীলনী ১৬.৩: যদি তুমি গেম অফ লাইফ বাস্তবায়ন করে থাক, তাহলে তুমি গ্রিড ওয়ার্ল্ড স্টুডেন্ট ম্যানুয়ালের পর্ব ৪ এর জন্য প্রস্তুত। পড় এবং অনুশীলন কর।

তোমাকে অভিবাদন, তুমি এটা শেষ করেছ!

পরিশিষ্ট A
গ্রাফিক্স (Graphics)

A.1 জাভা দ্বিমাত্রিক গ্রাফিক্স (Java 2D Graphics)

এই পরিশিষ্টতে জাভা গ্রাফিক্সে যে উদাহরণ এবং অনুশীলনীগুলো প্রদর্শিত হয়েছিল তা প্রদান করে। জাভায় বিভিন্ন উপায়ে গ্রাফিক্স তৈরি করা যায়; এর মাঝে সহজটি হল java.awt.Graphics এটা ব্যবহার করা। এখানে একটি উদাহরণ দেওয়া হল:

```
import java.awt.Canvas;
import java.awt.Graphics;
import javax.swing.JFrame;

public class MyCanvas extends Canvas {

	public static void main(String[] args) {
		// make the frame
		JFrame frame = new JFrame();
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		// add the canvas
		Canvas canvas = new MyCanvas();
		canvas.setSize(400, 400);
		frame.getContentPane().add(canvas);

		// show the frame
		frame.pack();
		frame.setVisible(true);
		}

		public void paint(Graphics g) {
		// draw a circle
		g.fillOval(100, 100, 200, 200);
		}
	}
```

তুমি এই কোডটি http://thinkapjava.com/code/MyCanvas.java এখান থেকে ডাউনলোড করতে পারবে।

java.awt এবং javax.swing থেকে আমাদের যে ক্লাসগুলো প্রয়োজন তা প্রথম লাইন ইমপোর্ট করে।

MyCanvas প্রসারিত করে Canvas কে, যার অর্থ হল যে একটি MyCanvas অবজেক্ট হল এক প্রকার Canvas যেটা গ্রাফিক্যাল অবজেক্ট আঁকতে প্রদান করা হয়।

main এ আমরা,

1. একটি Jframe তৈরি করতে পারি, যেটা একটি উইন্ডো যা ক্যানভাস, বাটন, মেনু এবং অন্যান্য উইন্ডো কম্পোনেন্টগুলোকে ধারণ করতে পারে;
2. একটি MyCanvas তৈরি কর, এটার উচ্চতা ও প্রস্থ নির্ধারণ কর, এবং এটি frame এ যুক্ত কর; এবং
3. স্ক্রিনে frame টি প্রদর্শন কর।

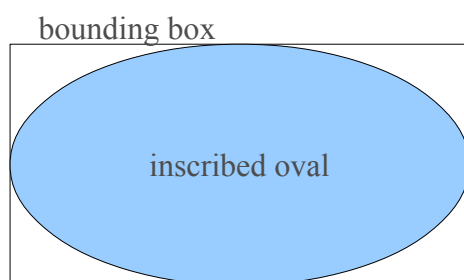
paint একটি বিশেষ মেথড যা MyCanvas যখন আঁকার দরকার হয় তখন যুক্ত হয়। তুমি যদি কোডটি রান করাও, তুমি একটি কাল বৃত্ত দেখবে গ্রে ব্র্যাক গ্রাউন্ডে।

A.2 গ্রাফিক্স মেথড (Graphics methods)

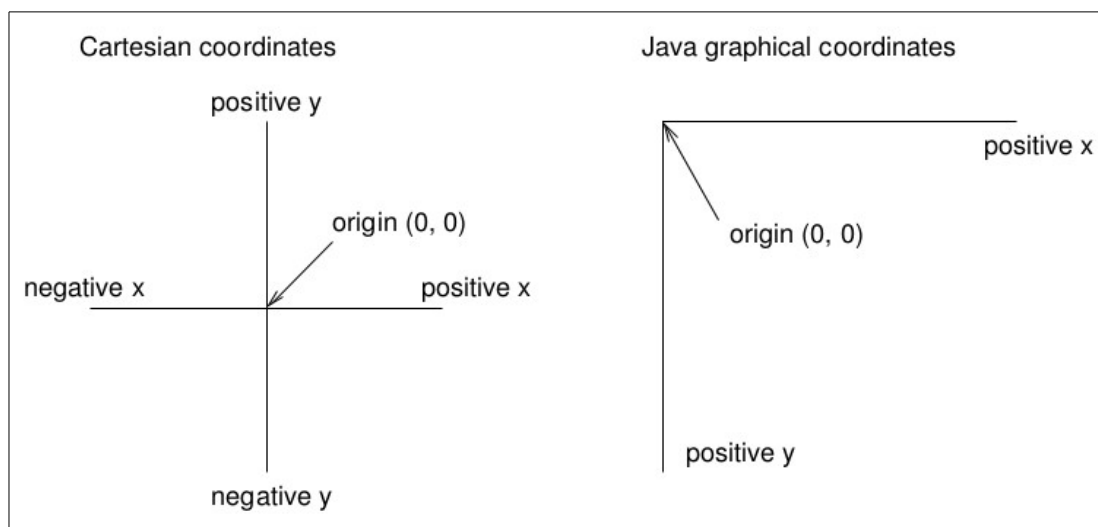
ক্যাম্পাসে আঁকতে চাইলে, তোমার গ্রাফিক্স অবজেক্টে মেথড যুক্ত করতে হবে। পূর্ববর্তী উদাহরণে fillOval ব্যবহার করা হয়েছিল। এছাড়াও drawLine, drawRect এবং আরও অনেক মেথড যুক্ত করা হয়েছিল। তুমি এই মেথডের ডকুমেন্টেশন এখান থেকে পড়তে পার,
http://download.oracle.com/javase/6/docs/api/java/awt/Graphics.html

এখানে fillOval এর প্রোটোটাইপ দেখানো হল:

```
public void fillOval(int x, int y, int width, int height)
```

প্যারামিটারটি একটি bounding box কে নির্দিষ্ট করে, যেটা আমরা যে অধিবৃত্ত একেছি তার ভেতর একটি আয়তক্ষেত্র (ছবিতে দেখানো হল)। bounding box নিজে নিজে অংকিত হয় না।

গ্রাফিক্স coordinate system এ x এবং y bounding box এর উপরের বাম কর্ণারকে নির্দেশ করে।

A.3 কর্ডিনেট (Coordinates)

তুমি সম্ভবত দ্বিমাত্রিক কার্টেসিয়ান কর্ডিনেটের সাথে পরিচিত, যেখানে প্রতিটা স্থান একটি x-অক্ষ এবং y- অক্ষে বিভক্ত। নিয়মানুযায়ী, কার্টেসিয়ান কর্ডিনেট শুধু মাত্র ডানদিকে এবং উপর দিকে বৃদ্ধি পায়, নিচের চিত্রে দেখান হল।

নিয়মানুযায়ী, কম্পিউটার গ্রাফিক্স সিস্টেম একটি কর্ডিনেট সিস্টেম ব্যবহার করে, যেটার শুরু হয় উপরের বাম কর্নার থেকে, এবং ধনাত্মক y-অক্ষ থাকে নিচের দিকে। জাভা এই নিয়ম মেনে চলে।

কর্ডিনেটগুলোর পরিমাপ করা হয় পিক্সেলে; প্রতিটি পিক্সেল স্ক্রিনের একটি ডটকে ধারণ করে। একটি আদর্শ স্ক্রিন প্রায় ১০০০ পিক্সেল চওড়া হয়। কর্ডিনেটগুলো সর্বদাই ইনটেজার সংখ্যা হয়। তুমি যদি একটি ফ্লোটিং-পয়েন্ট ভ্যালুকে কর্ডিনেট হিসাবে ব্যবহার করতে চাও, তোমার এটিকে পূর্ণ সংখ্যায় নিয়ে আসতে হবে (সেকশন ৩.২ দেখ)।

A.4 কালার (Color)

একটি আকৃতির জন্য কোন কালার পছন্দ করতে, গ্রাফিক্স অবজেক্টে setColor যুক্ত করতে হবে:

```
g.setColor(Color.red);
```

setColor বর্তমান কালারকে পরিবর্তন করে; এটি অঙ্কনে বর্তমান যে সব কালার পায় তার সবই পরিবর্তন করে।

Color.red একটি মান যেটা কালার ক্লাস দ্বারা সরবরাহ হয়; এটা ব্যবহার করতে java.awt.Color ইমপোর্ট করতে হবে। অন্য কালার যুক্ত করতে:

black	blue	cyan	darkGray	gray	lightGray
magenta	orange	pink	red	white	yellow

তুমি red, green এবং blue (RGB) এর কম্পোনেন্ট নির্দিষ্ট করে নতুন কোন কালার তৈরি করতে পার। বিস্তারিত এখানে দেখ http://download.oracle.com/javase/6/docs/api/java/awt/Color.html

তুমি Canvas.setBackground ব্যবহার করে ক্যানভাসের ব্যাকগ্রাউন্ড কালার নিয়ন্ত্রণ করতে পার।

A.5 মিকি মাউস (Mickey Mouse)

চল আমরা এখন বলি আমরা একটি মিকি মাউসের ছবি আঁকতে চাই। আমরা শুধু মুখ আঁকতে ওভাল ব্যবহার করতে পারি, এবং এরপর আমরা কান যুক্ত করতে পারি। এই কোডকে আরও বেশি রিডেবল করতে, আমরা bounding boxes এর পরিবর্তে আমরা আয়তক্ষেত্র ব্যবহার করতে পারি।

এখানে একটি মেথড দেখানো হল যেখানে একটি আয়তক্ষেত্র গ্রহণ করা হয়েছে এবং fillOval যুক্ত করা হয়েছে।

```
public void boxOval(Graphics g, Rectangle bb) {
        g.fillOval(bb.x, bb.y, bb.width, bb.height);
}
```

এবং এখানে মিকি আঁকার মেথড দেওয়া হল:

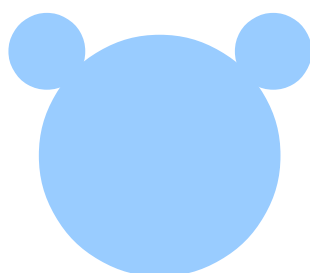
```
public void mickey(Graphics g, Rectangle bb) {
        boxOval(g, bb);

        int dx = bb.width/2;
        int dy = bb.height/2;
        Rectangle half = new Rectangle(bb.x, bb.y, dx, dy);

        half.translate(-dx/2, -dy/2);
        boxOval(g, half);

        half.translate(dx*2, 0);
        boxOval(g, half);
}
```

প্রথম লাইনটি মুখ আঁকে। পরের তিন লাইন কানের জন্য একটি ছোট আয়তক্ষেত্র আঁকে। আমরা আয়তক্ষেত্রকে উপরে এবং বাম কর্ণারে বসিয়েছি, এরপর দ্বিতীয়টি ডান দিকে বসিয়েছি।

ফলাফল হিসাবে নিচের চিত্রটা পেয়েছি:

তুমি এর কোডগুলো ডাউনলোড করতে পার এখান থেকে http://thinkapjava.com/code/Mickey.java.

A.6 শব্দকোষ (Glossary)

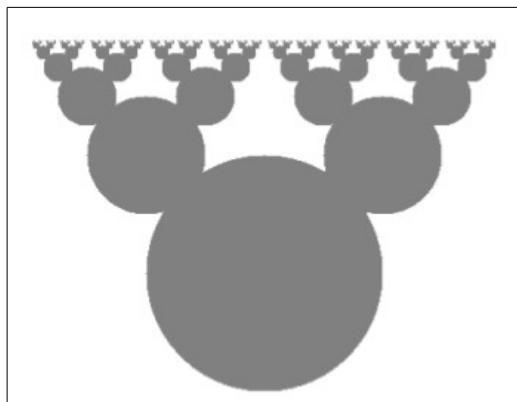
অক্ষাংশ(coordinate): একটি ভ্যারিয়েবল বা মান যেটা দ্বিমাত্রিক গ্রাফিক্যাল উইন্ডোর অবস্থানকে নির্দেশ করে।
পিক্সেল (pixel): অক্ষাংশ বা coordinate পরিমাপের একক।

বাউন্ডিং বক্স (bounding box): একটি আয়তক্ষেত্রের ক্ষেত্রকে নির্দিষ্ট করার একটি উপায়।

A.7 অনুশীলনী

অনুশীলনী A.1. জাপানের পতাকা আঁক, সাদা ব্র্যাকগ্রাউন্ডে একটি লাল বৃত্ত যেটা লম্বার তুলনায় চওড়ায় বড়।

অনুশীলনী A.2. Mickey.java কে পরিবর্তন করে কানের উপর কান আঁক, আবার এর উপর কান আঁকতে থাক যতক্ষণ পর্যন্ত না এটি তিন পিক্সেল চওড়া হয়।

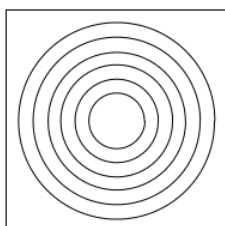
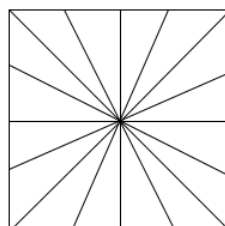
ফলাফল দেখতে এই রকম হবে:

ইঙ্গিত: তোমার শুধু মাত্র কোডের কিছু লাইনে যোগ করতে হবে অথবা পরিবর্তন করতে হবে।

তুমি এর একটি সমাধান এখান থেকে পাবে http://thinkapjava.com/code/MickeySoln.java.

অনুশীলনী A.3.
১। http://thinkapjava.com/code/Moire.java ডাউনলোড কর এবং এটি তোমার ডেভেলমেন্ট এনভায়রনমেন্ট এ ইমপোর্ট কর।

২। paint মেথডটি রিড কর এবং তোমার মত করে একটি স্কেচ আঁক। তুমি যা আঁকতে চাচ্ছ তাই কি আঁকতে পারছ? উদাহরণস্বরূপ এখানো দেখ http://en.wikipedia.org/wiki/Moire_pattern

৩। প্রোগ্রামের পরিবর্তন কর যেন বৃত্তের মাঝের স্পেস ছোট অথবা বড় হয়। ইমেজে কি হয় তা দেখ।

৪। প্রোগ্রামের পরিবর্তন কর যেন বৃত্তটি স্ক্রিনের মাঝখানে আঁকা হয় এবং সমকেন্দ্রীয় হয়, নিচের বামপাশের ছবির মত। বৃত্তগুলোর মাঝে দুরত্বটা হবে যথেষ্ট পরিমান ছোট যাতে Moir e Interference খুব স্পষ্ট হয়।

৫। radial নামে একটি মেথড রাইট কর যেটা চিত্রে দেখানো (ডানপাশে) লাইন সেগম্যান্টের মতো একটি রেডিয়াল সেট আঁকতে পারে, কিন্তু তারা খুব কাছে কাছে হবে যাতে Moire pattern তৈরি করতে পারে।

৬। যে কোন ধরনের গ্রাফিক্যাল প্যাটার্ন Moire-like interference প্যাটার্নে পরিবর্তিত হতে পারে। এখন নিজে করে দেখ আর কি কি তুমি তৈরি করতে পার।

পরিশিষ্ট B

জাভায় ইনপুট এবং আউটপুট (Input and Output in Java)

B.1 সিস্টেম অবজেক্ট (System objects)

সিস্টেম ক্লাস মেথড এবং অবজেক্ট প্রদান করে থাকে যেটা কীবোর্ড থেকে ইনপুট করা হয়, স্ক্রিনে টেক্সট প্রিন্ট করা হয়, এবং ফাইল ইনপুট এবং আউটপুট করা হয় (I/O)। System.out একটি অবজেক্ট যেটা স্ক্রিনে ডিসপ্লে করে। যখন তুমি print এবং println যুক্ত করবে, তখন তোমাকে এদের সাথে System.out ও যুক্ত করবে।

তুমি এমনকি System.out প্রিন্ট করতে System.out ও ব্যবহার করতে পারো:

```
System.out.println(System.out);
```

ফলাফল হবে:

```
java.io.PrintStream@80cc0e5
```

যখন জাভা একটি অবজেক্ট প্রিন্ট করে, এটা তখন অবজেক্টের (PrintStream) টাইপ প্রিন্ট করে, প্যাকেজের টাইপ যেখানে সংজ্ঞায়িত করা হয় (java.io), এবং এটি একটি অবজেক্ট আইডেন্টিফায়ারের জন্য অনন্য। আমার মেশিনের জন্য আইডেন্টিফায়ার হল 80cc0e5, কিন্তু তুমি যদি এই একই কোড রান কর তুমি সম্ভবত ভিন্ন কিছু পাবে।

System.in এই নামে একটি অবজেক্ট আছে যেটা কীবোর্ড থেকে ইনপুট নেওয়াটা সম্ভব করে তুলে। দুর্ভাগ্যক্রমে, কীবোর্ড থেকে ইনপুট নেওয়াটা সহজ করে না।

B.2 কীবোর্ড ইনপুট (Keyboard input)

প্রথমে, একটি নতুন InputStreamReader তৈরি করতে তোমাকে System.in ব্যবহার করতে হবে।

```
InputStreamReader in = new InputStreamReader(System.in);
```

এরপর একটি নতুন BufferedReader তৈরি করতে ব্যবহার করুন:

```
BufferedReader keyboard = new BufferedReader(in);
```

সর্বশেষে কীবোর্ড থেকে ইনপুট নিতে এবং এটিকে একটি স্টিং হিসাবে কনভার্ট করতে, তোমাকে keyboard এর readLine যুক্ত করতে হবে।

```
String s = keyboard.readLine();
System.out.println(s);
```

এখানে শুধু একটি সমস্যা আছে। এখানে কিছু জিনিস আছে যা ভুল জায়গায় নিতে পারে যখন তুমি readLine যুক্ত করবে এবং তারা সম্ভবত তোমাকে একটা IOException দিবে। একটি মেথড যা একটি এক্সপ্রেশন throw করে তা এখানে প্রটোটাইপ হিসাবে যুক্ত হতে পারে, যেমন:

```
public static void main(String[] args) throws IOException {
    // body of main
}
```

B.3 ফাইল ইনপুট (File input)

এখানে একটি প্রোগ্রাম দেওয়া হল যেটা ফাইল থেকে লাইন পড়তে পারে এবং প্রিন্ট করে:

```
import java.io.*;

public class Words {

    public static void main(String[] args)
        throws FileNotFoundException, IOException {

        processFile("words.txt");
    }

    public static void processFile(String filename)
        throws FileNotFoundException, IOException {

        FileReader fileReader = new FileReader(filename);
        BufferedReader in = new BufferedReader(fileReader);

        while (true) {
        String s = in.readLine();
        if (s == null) break;
        System.out.println(s);
```

```
        }
    }
}
```

প্রথম লাইন java.io ইমপোর্ট করে, প্যাকেজ যা FileReader, BufferedReader এবং জাভা সাধারণ কাজের জন্য যে বিস্তৃত ক্লাস হায়ারকি ধারণ করে। * এর অর্থ হল এটি প্যাকেজের মধ্যে উপস্থিত সকল ক্লাস ইমপোর্ট করে।

একই প্রোগ্রাম পাইথনে কিভাবে চলে তা দেখানো হল:

```
for word in open('words.txt'):
    print word
```

আমি কিডিং করছি না। এটাই সমগ্র প্রোগ্রাম, এবং এটা একই জিনিস।

B.4 ক্যাচিং এক্সসেপশন (Catching exceptions)

পূর্ববর্তী উদাহরণে দেখেছি, processFile FileNotFoundException এবং IOExceptio নিক্ষেপ করতে পারে। এবং যেহেতু main processFile কে কল করতে পারে, এটা একই এক্সসেপশনকে ডিক্লেয়ার করতে পারে। একটি বড় প্রোগ্রামে, main অবশ্যই প্রতিটি এক্সসেপশনে ডিক্লেয়ার করতে পারে।

বিকল্পটি হল try স্টেটমেন্টের সাথে এক্সসেপশন ধরে রাখা। এখানে একটি উদাহরণ দেওয়া হল:

```
    public static void main(String[] args) {
        try {
            processFile("words.txt");
        } catch (Exception ex) {
            System.out.println("That didn't work.     Here's why:");
            ex.printStackTrace();
        }
    }
```

এটার গঠন কিছুটা if স্টেটমেন্টের মতোই। যদি প্রথম “branch” এক্সসেপশন ছাড়া রান করতে পারে, তাহলে দ্বিতীয় branch স্কিপ করবে। যদি প্রথম branch একটি এক্সসেপশনের causes হয়, তাহলে এক্সিকিউশনের ধারা দ্বিতীয় branch এর দিকে যাবে, যেখানে এক্সসেপশনাল কন্ডিশনের সাথে ডিল করবে (সহজভাবে “error” প্রদানের মাধ্যমে)। এই ক্ষেত্রে একটি এরর মেসেজ দেখাবে এবং চিহ্ন স্তূপী করা থাকবে।

তুমি এই কোডটি http://thinkapjava.com/code/Words.java এখান থেকে ডাউনলোড করতে পারবে। এবং ওয়ার্ড লিস্ট পাবে এখানে http://thinkapjava.com/code/words.txt.

এখন উদাহরণ ৮.৯, ৮.১০ এবং ৮.১১ কর।

অ্যাপেনডিক্স C

প্রোগ্রাম ডেভেলপমেন্ট

C.1 স্ট্র্যাটেজি

আমি এই বইয়ে ভিন্ন ভিন্ন প্রোগ্রাম ডেভেলপমেন্ট স্ট্র্যাটেজি উপস্থাপন করেছি, সুতরাং আমি এগুলোকে এখানে একত্রিত করতে চাই। সকল স্ট্র্যাটেজির ভিত্তি হচ্ছে incremental development, যা এরকম:

১। এমন একটি প্রোগ্রাম দ্বারা আরম্ভ কর যা দৃশ্যমান কিছু করে, যেমন কোন কিছু প্রিন্ট করা।
২। এক এক করে ছোট ছোট লাইনের কোড যোগ কর, এবং প্রতিটি পরিবর্তনের পর প্রোগ্রাম পরীক্ষা কর।
৩। এই পরীক্ষা চালিয়া যাও এবং পুরাবৃত্তি কর যতক্ষন না এটা সঠিক কাজ করতে পারছে।

প্রতিটি পরিবর্তনের পর, প্রোগ্রাম অবশ্যই একটি দৃশ্যমান আবহ তৈরি করবে যা নতুন কোড পরীক্ষা করবে। প্রোগ্রামিং এ এভাবে আগালে অনেক সময় বাঁচবে।

যেহেতু তুমি প্রতি মুহূর্তে কিছু সংখ্যক কোড যোগ করছ, সুতরাং ক্রুটি খোঁজার জন্য এটা উত্তম। এবং যেহেতু প্রোগ্রামের প্রতিটি সংস্করণ একটি দৃশ্যমান ফলাফল তৈরি করে, সুতরাং প্রতি মুহূর্তে তুমি নিজে নিজে প্রোগ্রামের একটি কাল্পনিক মডেল তৈরি করে নিতে পারছ। যদি সেখানে মনে কর ভুল হবে (এবং সঠিক করার মত সুযোগ পাবে) তাহলে ভুল কোড লিখার পূর্বেই তুমি থামতে পারবে।

ক্রমবর্ধমান উন্নতির চ্যালেঞ্জ হচ্ছে যে প্রোগ্রামের ক্রুটি ঠিক করার সময় কোথা হতে শুরু করতে হবে তা খুঁজে বের করা। এটা সহজ করার জন্য কিছু স্ট্র্যাটেজি রয়েছে:

এনক্যাপসুলেশন এবং সর্বজনীনকরণ: তুমি যদি এখনও না জেনে থাক কিভাবে কম্পিউটেশনকে মেথডে ভাগ করা হয়, main এ কোড লিখা শুরু কর, তারপর সুসঙ্গত জায়গায় মেথড এনক্যাপসুলেট কর, এবং উপযুক্তভাবে সর্বজনীন কর।

র‍্যাপিড প্রোটোটাইপিং: যদি তুমি জান কোন মেথডটি লিখতে হবে, কিন্তু কিভাবে লিখতে হবে সেটা না জান, একটি সাধারণ কেস এর রাফ ড্রাফট লিখা শুরু কর, তারপর অন্যান্য কেস এর সাথে তা পরীক্ষা কর, এভাবে আগাতে থাক এবং প্রোগ্রাম ঠিক করতে থাক।

বটম-আপ: সাধারণ মেথড লিখার মাধ্যমে শুরু কর, তারপর একটি সমাধানের মাধ্যমে এর সুরাহা কর।

টপ-ডাউন: কম্পিউটেশনের স্ট্রাকচার ডিজাইন এবং যেসকল মেথড তোমার প্রয়োজন তা নির্ণয় করতে সুডোকোড ব্যবহার কর। তারপর মেথড লিখ এবং সুডোকোড কে বাস্তব কোড দ্বারা প্রতিস্থাপন কর।

একই সাথে, তোমার স্ক্যাফোল্ড করার প্রয়োজন হতে পারে। উদাহরণসরূপ, প্রতিটি ক্লাসের একটি toString মেথড থাকা উচিত, যা তোমাকে অবজেক্টের অবস্থা মনুষ্য পঠন উপযোগী করে প্রিন্ট করতে দিবে। এই মেথডটি ডিবাগ করার জন্য উপকারী, কিন্তু সাধারণত এটি একটি সম্পূর্ণ প্রোগ্রামের অংশ নয়।

C.2 ফেইলিউর মোড

তুমি যদি ডিবাগিং এর জন্য প্রচুর সময় ব্যয় করে থাক, এর কারন সম্ভবত তুমি ডেভেলপমেন্টের জন্য একটি অকার্যকর স্ট্র্যাটেজি ব্যবহার করছ। নিচে কিছু ফেইলিউর মোড দেয়া হল যা আমি প্রায়শই দেখি:

Non-incremental ডেভেলপমেন্ট: যদি তুমি কম্পাইল এবং পরীক্ষা না করে কয়েকটি লাইনের বেশী কোড লিখ, তাহলে তুমি বিপদকে ডাকছ। একসময় আমি এক ছাত্রকে জিজ্ঞাসা করেছিলাম তোমার হোমওয়ার্ক কেমন যাচ্ছে, সে বলল "চমৎকার! আমার সব লিখাই ছিল। এখন শুধু আমাকে সেটা ডিবাগ করতে হবে।"

খারাপ কোডের সংযুক্তি: যদি তুমি কম্পাইল না করে এবং পরীক্ষা না করে কিছু লাইনের বেশী কোড লিখ, তুমি হয়ত সেটিকে আর ডিবাগ করতে পারবে না। মাঝে মধ্যে শুধু মাত্র কোড মুছে নতুন করে লিখাই একমাত্র স্ট্র্যাটেজি। কিন্তু শিক্ষানবিস রা প্রায়ই তাদের কোডের উপর আবেগপ্রবণ হয়ে যায়, কোড কাজ করুন আর নাই করুক। এটাকে কাটিয়ে উঠতে হলে একটু নির্মম হতেই হবে।

Random-walk প্রোগ্রামিং: আমি মাঝে মধ্যে কিছু ছাত্রের সাথে কাজ করি যারা র‍্যানডম ভাবে প্রোগ্রাম করে থাকে। তারা পরিবর্তন করে, প্রোগ্রাম চালায়, ক্রুটি পায়, পরিবর্তন করে, প্রোগ্রাম চালায়, ইত্যাদি। সমস্যা হচ্ছে আপাতদৃষ্টিতে প্রোগ্রামের ফলাফল এবং পরিবর্তনের মাঝে কোন সংযোগ নেই। যদি তুমি ক্রুটির বার্তা পাও, সেটা পড়ার জন্য সময় নাও। আরও সাধারণ ভাবে বললে, চিন্তা করার জন্য সময় নাও।

কম্পাইলার সাবমিশন: ক্রুটির বার্তা উপকারী, কিন্তু তারা সবসময় সঠিক নয়, উদাহরণসরূপ, যদি বার্তাটি বলে, "Semi-colon expected on line 13," এর মানে হল লাইন ১৩ এর নিকট একটি সিনট্যাক্স ক্রুটি রয়েছে। কিন্তু লাইন ১৩তে একটি সেমিকোলন দিয়ে দিলেই এর সমাধান হবে না। কম্পাইলারের ইচ্ছার বিরুদ্ধে কাজ করো না।

পরবর্তী অধ্যায় কার্যকরী ডিবাগিং এর জন্য আরও সাজেশন তৈরি করে।

অ্যাপেনডিক্স D

ডিবাগিং

শ্রেষ্ট ডিবাগিং স্ট্র্যাটেজি নির্ভর করে তোমার কি ধরনের ত্রুটি রয়েছে তার উপর:

সিনট্যাক্স ত্রুটি তৈরি হয় কম্পাইলারের মাধ্যমে এবং নির্দেশিত করে প্রোগ্রামের সিনট্যাক্সে ত্রুটি রয়েছে। উদাহরণ: একটি স্টেটমেন্টের শেষে সেমিকোলন বাদ দিয়ে যাওয়া।

ব্যাতিক্রম ঘটে যখন প্রোগ্রাম চলার সময় কিছু একটা ত্রুটি দেখা যায়। উদাহরণ: একটি অসীম রিকারশন একটি StackOverflowException উৎপন্ন করে।

লজিক ত্রুটি প্রোগ্রামকে ভুল জিনিস করতে বলে। উদাহরণ: একটি এক্সপ্রেশন তুমি যেভাবে চাও সেভাবে মূল্যায়ন করা নাও হতে পারে।, ব্যাতিক্রম ফলাফল মেনে নিতে হবে।

নিচের অংশ গঠিত হয়েছে ত্রুটির ধরনের উপর; কিছু টেকনিক একের অধিক ধরনের জন্য প্রযোজ্য।

D.1 সিনট্যাক্স ত্রুটি

উত্তম ধরনের ডিবাগিং হল কিছুই না করা কারন প্রথম থেকেই যেন ত্রুটি না হয় সেদিকে লক্ষ রেখে প্রোগ্রাম করা। পূর্ববর্তী অংশে, আমি ডেভেলপমেন্ট স্ট্র্যাটেজি সাজেষ্ট করেছি যা ত্রুটি কমিয়ে আনে এবং ত্রুটি থাকলে তা সহজেই খুঁজে বের করা যায়। আসল ব্যপার টা হচ্ছে একটি কার্যকর প্রোগ্রাম নিয়ে কাজ শুরু করা এবং একই সময়ে ছোট ছোট কোড সংযুক্ত করে এগিয়ে যাওয়া। যখন একটি ত্রুটি ধরা পরবে, তুমি তখন বুঝতে পারবে কোথায় এই ঘটনা ঘটেছে।

তারপরেও, তুমি নিজেকে নিচের যে কোন অবস্থায় আবিষ্কার করতে পার। প্রতিটি অবস্থায় কিভাবে এগিয়ে যেতে হবে সেটা নিয়ে আমি আলোচনা করেছি।

কম্পাইলার বহু সংখ্যক ত্রুটির বার্তা প্রদর্শন করছে।

যদি কম্পাইলার ১০০ ত্রুটির বার্তা প্রদর্শন করে এর মানে এই নয় যে তোমার প্রোগ্রামে ১০০ ত্রুটি রয়েছে। যখন কম্পাইলার ত্রুটির সম্মুখীন হয়, এটা প্রায়শই খেই হারিয়ে ফেলে। প্রথম ত্রুটি চিহ্নিত করার পর এটি ত্রুটি মুক্ত হওয়ার চেষ্টা করে, কিন্তু মাঝে মাঝে এটি ভুল ত্রুটি প্রদর্শন করে।

শুধুমাত্র প্রথম ত্রুটির বার্তা সত্যিকার অর্থে নির্ভরযোগ্য। আমি সাজেষ্ট করব তুমি একক সময়ে একটি করে ত্রুটির সমাধান করবে এবং পুনরায় প্রোগ্রাম কম্পাইল করবে। এমনও হতে পারে, শুধুমাত্র একটি সেমিকোলন ১০০ টি ত্রুটি দূর করে দিয়েছে।

আমি কম্পাইলার হতে একটি অদ্ভূত বার্তা পাচ্ছি এবং কিছুতেই এটি দূর হচ্ছে না।

প্রথমত, ত্রুটির বার্তাটি ভাল করে পড়। এটি বাহুল্যবর্জিত অপভাষায় লিখা রয়েছে, কিন্তু খেয়াল করলে তুমি এর

ভেতর লুকানো কার্নেলের তথ্য খুঁজে পাবে।

যদি তা না হয়, বার্তাটি তোমাকে বলবে প্রোগ্রামের কোথায় ত্রুটির ঘটেছে। আসলে, এটি তোমাকে বলবে ত্রুটি ঘটার সময় কম্পাইলার কোথায় ছিল, যা আসলে ত্রুটি কোথায় সেটা বলবে না। কম্পাইলার তোমাকে যে তথ্য দিচ্ছে তা গাইডলাইন হিসেবে ব্যবহার কর, কিন্তু তুমি যদি কোন ত্রুটি দেখতে না পাও যেটা কম্পাইলার তোমাকে দেখাতে চাচ্ছে, আরও গভীরভাবে অনুসন্ধান কর।

সাধারণত প্রোগ্রামে ত্রুটি বার্তার পূর্বে কোন জায়গায় ত্রুটি ঘটে থাকে, কিন্তু এমন অনেক সময় রয়েছে যে সম্পূর্ণ ভিন্ন জায়গায় ত্রুটি ঘটে থাকে। উদাহরণস্বরূপ, তুমি যদি মেথড ইনভোকেশনে ত্রুটি পাও তাহলে আসল ত্রুটি ঘটেছে মেথড ডেফিনিশনে।

যদি তুমি দ্রুত ত্রুটি খুঁজে না পাও, একটি বিশ্রাম নাও এবং পুরো প্রোগ্রামে পুনরায় খুঁজা শুরু কর। নিশ্চত হও যে প্রোগ্রামটি ঠিক মত ইনডেন্ট করা হয়েছে কিনা; যা সিনট্যাক্স ত্রুটি ধরতে সহায়তা করে।

এখন, সাধারণ ত্রুটি খুঁজে বের কর:

১। পরীক্ষা কর, সকল রকমের বন্ধনীর ভারসাম্য রয়েছে কিনা এবং ঠিক মত নেস্টেড রয়েছে কিনা। সকল মেথড ডেফিনিশন অবশ্যই একটি ক্লাসের মধ্যে নেস্টেড অবস্থায় থাকতে হবে। সকল প্রোগ্রাম স্টেটমেন্ট একটি মেথড ডেফিনিশনের মধ্যে থাকতে হবে।

২। মনে রাখবে ছোট হাতের অক্ষর এবং বড় হাতের অক্ষর একই জিনিস নয়।

৩। স্টেটমেন্টের শেষে সেমিকোলন আছে নাকি তা পরীক্ষা কর। (এবং squiggly-braces এর পর কোন সেমিকোন হবে না)।

৪। নিশ্চিত হও কোডের যে কোন স্ট্রিং এ একই রকমের কোটেশন চিহ্ন রয়েছে কিনা। নিশ্চিত হও যে স্ট্রিং এর জন্য ডাবল কোটেশন এবং অক্ষরের জন্য সিঙ্গেল কোটেশন ব্যবহার করেছ।

৫। সকল এসাইনমেন্ট স্টেটমেন্টে নিশ্চিত হও বাম পাশের ধরন এবং ডান পাশের ধরন একই রকমের। নিশ্চিত হও বাম পাশের এক্সপ্রেশন হচ্ছে একটি ভ্যারিয়েবলের নাম অথবা অন্য কিছু যার মান তুমি নির্ধারণ করবে (অ্যারে এর উপাদান এর মত)।

৬। সকল মেথড ইনভোকেশনের জন্য, নিশ্চিত হও তুমি যে আর্গুমেন্ট প্রদান করছ তা সঠিক ক্রমে রয়েছে, এবং সঠিক ধরনে রয়েছে, এবং যে অবজেক্ট তুমি ইনভোক করছ তা সঠিক ধরনের।

৭। যদি তুমি ভ্যালু মেথড ইনভোক কর, নিশ্চিত হও তুমি ফলাফলের সাথে কিছু করছ। যদি তুমি ভয়েড মেথড ইনভোক কর, নিশ্চিত হও তুমি ফলাফলের সাথে কিছু করার চেষ্টা করছ না।

৮। যদি তুমি অবজেক্ট মেথড ইনভোক কর, নিশ্চিত হও তুমি এমন অবজেক্ট ইনভোক করছ যার ধরন সঠিক। যদি তুমি বাইরে থেকে ক্লাস মেথড ইনভোক কর যেখানে এটি ডিফাইন করা রয়েছে, নিশ্চিত হও তুমি ক্লাসের নাম বলছ কিনা।

৯। অবজেক্ট মেথডের ভেতর তুমি ইনসট্যান্স ভ্যারিয়েবলে রেফার করতে পার কোন অবজেক্ট নির্ধারণ না করে। যদি তুমি একটি ক্লাস মেথডে এটি কর, তুমি একটি বার্তা পাবে যা এরকম, "Static reference to non-static variable"।

যদি কিছুই কাজ না করে, পরবর্তী অংশে যাও...

আমি যত চেষ্টাই করি না কেন আমার প্রোগ্রাম কম্পাইল করতে পারছি না।

যদি কম্পাইলার তোমাকে বলে একটি ত্রুট ঘটেছে কিন্তু তুমি সেটা দেখতে না পাও, এর কারন হতে পারে তুমি এবং কম্পাইলার একই কোড দেখছ না। এক্ষেত্রে তোমার ডেভেলপমেন্ট এনভায়রনমেন্ট পরীক্ষা কর কম্পাইলার ঠিক মত বর্তমান প্রোগ্রামটিকে ফোকাস করে রয়েছে কিনা। যদি তুমি নিশ্চিত না হও প্রোগ্রামের উপরে নিশ্চত ভাবে ত্রুটিযুক্ত এমন একটি কোড লিখ। এবার পুনরায় কম্পাইল কর। যদি কম্পাইলার নতুন ত্রুটি খুঁজে না পায় তাহলে সম্ভবত তোমার ডেভেলপমেন্ট এনভায়রনমেন্টের অপশনে কিছু সমস্যা হয়েছে।

যদি তুমি সম্পূর্ণভাবে কোড পরীক্ষা করে থাক, এবং তুমি নিশ্চিত থাক যে, কম্পাইলার সঠিক ভাবে কোড কম্পাইল করছে, তাহলে এখনই সময় **debugging by bisection** এর।

তুমি যে ফাইল নিয়ে কাজ করছ তা অনুলিপি কর। যদি তুমি Bob.java নিয়ে কাজ কর, একটি অনুলিপি কর এবং নাম দাও Bob.java.old.

Bob.java এর প্রায় অর্ধেক কোড মুছে ফেল এবং কম্পাইল কর।

- যদি প্রোগ্রাম এখন কম্পাইল হয় তুমি জানবে যে ত্রুটি মুছে ফেলা অংশে ছিল। মুছে ফেলা কোড এনে জোড়া দাও এবং এভাবে পরীক্ষা চালাতে থাক।
- যদি প্রোগ্রাম এর পরেও কাজ না করে তাহলে এই অংশেই ত্রুটি রয়েছে। কোডের অর্ধেক মুছে ফেল এবং পরীক্ষা চালিয়ে যাও।

যখন তুমি ত্রুটি খুঁজে পাবে এবং সমাধান করতে পারবে, কোডের যে অংশ মুছে ফেলেছিলে তা এনে জোড়া দাও, একক সময়ে কিছু অংশ করে।

এই উপায়টি খুব বিচ্ছিরি, তবে এটি অনেক দ্রুত কাজ করবে এবং এটি অনেক বেশী নির্ভরযোগ্য।

কম্পাইলার আমাকে যা করতে বলেছে তা করেছি, কিন্তু তারপরেও এটি কাজ করছে না।

কিছু কম্পাইলার বার্তায় উপদেশ থাকে যেমন, "class Golfer must be declared abstract. It does not define int compareTo(java.lang.Object) from interface java.lang.Comparable." মনে হচ্ছে কম্পাইলার তোমাকে বলছে Golfer ডিক্লেয়ার করতে অ্যাবসট্রাক্ট ক্লাস হিসেবে, এবং তুমি যদি এই বই পড়ে থাক, তাহলে সম্ভবত এটা কি জিনিস এবং কিভাবে করতে হয় তা জানবে না।

সৌভাগ্যবশত কম্পাইলার এখানে ভুল করছে। এক্ষেত্রে সমাধান হচ্ছে নিশ্চিত করা যে Golfer এর একটি মেথড রয়েছে যার নাম compareTo যা একটি অবজেক্ট গ্রহন করে প্যারামিটার হিসেবে।

কম্পাইলারকে কখনও তোমাকে নেতৃত্ব করতে দিও না। ত্রুটির বার্তা তোমাকে প্রমান দেয় যে কিছু একটা গন্ডগোল রয়েছে, কিন্তু যে সমাধান তারা দেয় সেটা অনির্ভরযোগ্য।

D.2 রান টাইম ত্রুটি

আমার প্রোগ্রাম হ্যাং করছে।

যদি একটি প্রোগ্রাম থেমে যায় এবং কোন কাজ করবে না বলে মনে হয়, আমরা এটাকে বলি হ্যাং হয়ে যাওয়া। প্রায়ই এটা মনে হয় যে প্রোগ্রামটি একটি অসীম লুপে পড়েছে অথবা একটি অসীম রিকারশনে পড়েছে। যদি কোন নির্দিষ্ট লুপ কে তুমি সন্দেহ কর যে এটিই সমস্যার সৃষ্টি করছে, তৎক্ষণাৎ "entering the loop" বলার পূর্বেই একটি প্রিন্ট স্টেটমেন্ট যোগ কর এবং আরেকটি যোগ কর "exiting the loop" এর পরে।

প্রোগ্রাম চালনা কর। যদি তুমি প্রথম বার্তা পাও কিন্তু ২য় বার্তা না পাও, তাহলে তুমি অসীম লুপে পড়েছ। "Infinite loop" এ চলে যাও।

বেশীরভাগ ক্ষেত্রেই একটি অসীম রিকারশন একটি প্রোগ্রামকে কিছু সময়ের জন্য চালায় এবং তারপর একটি StacOverflowException তৈরি করে। যদি এটি ঘটে থাকে, তাহলে ২য় টাইটেল "Infinite recursion" এ যাও।

যদি তুমি StackOverflowException না পাও, কিন্তু তুমি ধারণা কর যে রিকারসিভ মেথডেই ত্রুটি রয়েছে, তখনও তুমি অসীম রিকারশন অংশে টেকনিক খাটাতে পার।

যদি এই সাজেশনগুলোর কোনটিই তোমাকে সহায়তা না করে, তুমি সম্ভবত তোমার প্রোগ্রামের এক্সিকিউট হওয়ার ফ্লো বুঝতে পারছ না। "Flow of execution" টাইটেলে যাও।

অসীম লুপ

যদি তুমি মনে কর তোমার একটি অসীম লুপ রয়েছে এবং তুমি নিশ্চিতভাবে জান যে সেই লুপ কোনটি, তাহলে লুপের শেষে একটি প্রিন্ট স্টেটমেন্ট যোগ কর যা ভ্যারিয়েবলের মান এবং শর্ত প্রিন্ট করবে।

উদাহরনস্বরূপ,

```
while (x > 0 && y < 0) {
            // do something to x
            // do something to y

      System.out.println("x: " + x);
      System.out.println("y: " + y);
      System.out.println("condition: " + (x > 0 && y < 0));
}
```

এখন তুমি প্রোগ্রাম চালনার সময় তিন লাইনের আউটপুট দেখবে প্রতিটি ক্ষেত্রে। লুপের শেষ সময়ে শর্ত অবশ্যই মিথ্যা হবে। যদি লুপ চলতেই থাকে, তুমি x এবং y এর মান দেখতে পাবে এবং ধরতে পারবে কেন তারা সঠিকভাবে আপডেট হয়নি।

অসীম রিকারশন

বেশীরভাগ ক্ষেত্রে একটি অসীম রিকারশন প্রোগ্রামের StackOverflowException হওয়ার কারন। কিন্তু প্রোগ্রাম যদি ধীর গতির হয় তাহলে স্ট্যাক পূর্ণ করতে প্রচুর সময় প্রয়োজন হবে।

যদি তুমি জান কোন মেথডটি অসীম রিকারশন সৃষ্টি করছে, পরীক্ষা কর সেখানে একটি মূল কারন রয়েছে। এখানে কিছু শর্ত রয়েছে যা মেথডকে একটি রিকারসিভ ইনভোকেশন করা ব্যাতীত রিটার্ন করে। যদি তা না হয়, তোমাকে পুনরায় অ্যালগরিদম নিয়ে চিন্তা করতে হবে এবং মূল কারন বের করতে হবে।

যদি একটি মূল কারন থাকে কিন্তু প্রোগ্রাম সেটাতে পৌছতে না পারে, মেথডের পূর্বে একটি প্রিন্ট স্টেটমেন্ট যোগ কর যা প্যারামিটার প্রিন্ট করবে। এখন তুমি প্রোগ্রাম চালনা করলে যতবার মেথড ইনভোক হবে ততবার কিছু সংখ্যক লাইনের আউটপুট দেখতে পাবে, এবং তুমি প্যারামিটারের মানও দেখতে পাবে। যদি প্যারামিটার বেস কেসের দিকে না যায়, তুমি হয়ত দেখতে পাবে সেটা কেন হচ্ছে।

এক্সিকিউশনের ফ্লো

তুমি যদি নিশ্চিতভাবে না জান কিভাবে এক্সিকিউশনের ফ্লো তোমার প্রোগ্রামের দিকে আসছে, প্রতিটি মেথডের পূর্বে একটি প্রিন্ট স্টেটমেন্ট যোগ কর "entering method foo” বার্তা লিখে, যেখানে "foo” হচ্ছে মেথডের নাম।

এখন তুমি প্রোগ্রাম রান করলে যেহেতু ইনভোক হয়েছে তাই সকল মেথডের ট্রেস প্রিন্ট করবে। তুমি চাইলে প্রতিটি সংগ্রহীত বার্তার আর্গুমেন্ট প্রিন্ট করতে পারবে। যখন তুমি প্রোগ্রাম চালাবে, পরীক্ষা কর যে মান গুলো সঠিক কিনা, এবং একটি খুব সাধারণ ত্রুটি পরীক্ষা কর—ভুল ক্রমানুসারে আর্গুমেন্ট প্রদান করা।

যখন আমি প্রোগ্রাম চালাই আমি এক্সেপশন পাই।

যখন একটি এক্সেপশন ঘটে, জাভা একটি বার্তা প্রিন্ট করে যার মধ্যে এক্সেপশনটির নাম লিখা থাকে, যেই লাইনে সমস্যা হয়েছে সেটি থাকে এবং একটি স্ট্যাক ট্রেস থাকে। স্ট্যাক ট্রেসে থাকে যে মেথডটি চলছে সেটি, যে মেথড এটিকে ইনভোক করেছে, যে মেথড অন্যটিকে ইনভোক করেছে ইত্যাদি।

প্রথম ধাপ হচ্ছে প্রোগ্রামটি পরীক্ষা করে দেখা কোথায় সমস্যা হয়েছে এবং কি ঘটেছে সেটা বুঝা।

NullPointerException: তুমি একটি ইনসট্যান্স ভ্যারিয়েবল একসেস করতে চেয়েছ অথবা অবজেক্টের একটি মেথডকে ইনভোক করতে চেয়েছ যা বর্তমানে null রয়েছে। তোমার উচিত খুঁজে বের করা কোন মেথডটি null এবং তারপর খুঁজা উচিৎ কিভাবে সেটা হয়েছে।

মনে রাখবে যখন তুমি অবজেক্ট টাইপের সাথে একটি ভ্যারিয়েবল ডিক্লেয়ার কর, এটা প্রাথমিক অবস্থায় null থাকে যতক্ষন পর্যন্ত তুমি কোন মান নির্দিষ্ট করে না দিচ্ছ। উদাহরণস্বরূপ, এই কোড একটি NullPointException সৃষ্টি করে:

```
Point blank;
System.out.println(blank.x);
```

ArrayIndexOutOfBoundsException: তুমি যেই ইনডেক্স অ্যারে অ্যাকসেস করার জন্য ব্যবহার করছ

সেটা হয় নেগেটিভ অথবা array.length-1 এর থেকে বৃহত্তর। তুমি যদি সমস্যা কোথায় সেটার কোন চিহ্ন পাও, তৎক্ষণাত একটি প্রিন্ট স্টেটমেন্ট যোগ কর ইনডেক্সের মান এবং অ্যারে এর দৈর্ঘ্য প্রিন্ট হওয়ার পূর্বেই। অ্যারেটি কি সঠিক আকৃতির? ইনডেক্সের মান কি সঠিক?

এবার প্রোগ্রামের উল্টোদিক থেকে কাজ কর এবং দেখ কোথা হতে অ্যারে এবং ইনডেক্স এসেছে। নিকটতম অ্যাসাইমেন্ট স্টেটমেন্ট খুঁজ এবং দেখ এটি সঠিক কাজ করছে কিনা।

যদি এদের কোনটির একটি প্যারামিটার হয়, যে স্থান হতে ইনভোক করা হয়েছে সেখানে যাও এবং দেখ মানগুলো কোথা হতে আসছে।

StackOverFlowException: "Infinite recursion" দেখ।

FileNotFoundException: এর মানে হচ্ছে জাভা যে ফাইলটি খুঁজছিল সেটি পায়নি। তুমি যদি প্রজেক্ট ভিত্তিক ডেভেলপমেন্ট এনভায়রনমেন্ট যেমন Eclipse ব্যবহার করে থাক, তোমাকে প্রজেক্টের ভেতর ফাইল ইমপোর্ট করতে হতে পারে। অন্যথায় নিশ্চিত হয়ে নাও ফাইলটি বিদ্যমান কিনা এবং রাস্তা সঠিক কিনা। এই সমস্যাটি তোমার ফাইল সিস্টেমের উপর নির্ভর করে সুতরাং এটিকে ট্র্যাকডাউন করা কঠিন।

ArithmeticException: এটা ঘটে যখন অ্যারিথমেটিক অপারেশনের সময় কিছু গন্ডগোল দেখা দেয়, বেশীরভাগ ক্ষেত্রেই শূণ্য দ্বারা ভাগ করার কারনে।

আমি অধিক সংখ্যক প্রিন্ট স্টেটমেন্ট যোগ করেছি আমি আউটপুটের বন্যায় ভেসে যাচ্ছি।
ডিবাগিং এর জন্য প্রিন্ট স্টেটমেন্ট ব্যবহার করার একটি সমস্যা হচ্ছে তুমি আউটপুটের চাপে সমাধিস্ত হতে পার। দুটি উপায় রয়েছে এগিয়ে যাবার জন্য: হয় আউটপুট ছোট করে ফেল অথবা প্রোগ্রাম ছোট করে আন।

আউটপুট সহজতর করার জন্য যে সকল প্রিন্ট স্টেটমেন্ট কোন কাজে লাগছে না সেগুলো কমেন্ট আউট কর অথবা মুছে ফেল। যেহেতু তুমি প্রোগ্রাম ডেভেলপ করছ সুতরাং তোমার প্রোগ্রামের কোড যেন সংক্ষিপ্ত, তথ্যবহুল এবং প্রোগ্রাম কি করছে তা দৃষ্টিগোচর হয়।

প্রোগ্রাম সহজতর করার জন্য প্রোগ্রাম যে সমস্যা নিয়ে কাজ করছে সেখানে আনুপাতিক ভাবে কমিয়ে নিয়ে আস। উদাহরণস্বরূপ, তুমি যদি একটি অ্যারে সর্ট করে থাক, ছোট অ্যারে সর্ট কর। প্রোগ্রাম যদি ব্যবহারকারীর কাছ থেকে ইনপুট নিয়ে থাকে, ত্রুটি সৃষ্টি করে এমন একটি সাধারণ ইনপুট প্রদান করার ব্যবস্থা কর।

সেই সাথে কোড পরিষ্কার কর। যেসকল কোডের কোন কাজ নেই তা সরিয়ে ফেল এবং যাতে সহজ ভাবে পড়া যায় সে ব্যবস্থা কর। উদাহরনস্বরূপ, তুমি যদি মনে করে থাক গভীর নেস্টেড অংশে প্রোগ্রামের ত্রুটি রয়েছে, সেই অংশটুকু সহজ করে পুনরায় লিখ। তুমি যদি একটি বৃহৎ মেথড কে সন্দেহ করে থাক, তাকে ছোট মেথডে ভাগ করে নিয়ে আস এবং আলাদা আলাদা করে পরীক্ষা কর।

যতসামান্য টেস্ট কেস খুঁজে বের করার পদ্ধতিও মাঝে মাঝে বাগ ধরতে সহায়তা করে থাকে। উদাহরনস্বরূপ, তুমি যদি বুঝতে পার প্রোগ্রামটির অ্যারেতে যখন জোড় সংখ্যা থাকে তখন কাজ করে, কিন্তু বেজোড় সংখ্যা থাকলে কাজ করে না, এটা তোমাকে কি ঘটছে সেটার একটা ইঙ্গিত দিবে।

প্রোগ্রাম বুঝতে পারলে তুমি অতি সূক্ষ বাগ ধরতে পারবে। তুমি যদি এমন কিছু পরিবর্তন কর এবং মনে কর এটা

প্রোগ্রামকে প্রভাবিত করবে না, এটা না করে দেখতে পার।

D.3. লজিক ক্রুটি

আমার প্রোগ্রাম কাজ করে না।

লজিক ক্রুটি খুঁজে পাওয়া দূরূহ কারন কম্পাইলার এবং রানটাইম সিস্টেম. সমস্যা সংক্রান্ত কোন তথ্য প্রদান করে না। শুধুমাত্র তুমি জান যে প্রোগ্রামটি কি করবে, এবং শুধুমাত্র তুমিই জান যে প্রোগ্রামটি এই কাজ করছে না। প্রথম ধাপ হচ্ছে কোড এবং যে ধরনের বৈশিষ্ট্য আমি পাচ্ছি তার মধ্যে সম্পর্ক ঘটানো। তোমার অনুমান করতে হবে প্রোগ্রামটি আসলে কি করছে। নিচে কিছু প্রশ্ন দেয়া হল যা তোমরা নিজেদের জিজ্ঞাসা করতে পারবে:

প্রোগ্রামের এমন কিছু কি রয়েছে যেটা করার কথা ছিল কিন করেনি? এই কাজটা যে ফাংশনের করার কথা ছিল তা খুঁজে বের কর এবং এটা যখন কাজ করার কথা তখন করছে কিনা পরীক্ষা কর। উপরের "Flow of execution" অংশ দেখ।

এমন কিছু কি হচ্ছে যেটা হওয়া উচিত নয়? কোডের ভেতর সেই অংশটি খুঁজে বের কর এবং দেখ এটা যখন কাজ করার কথা নয় তখন কাজ করছে নাকি।

কোডের কোন অংশ কি অপ্রত্যাশিত আবহ তৈরি করছে? নিশ্চিত হও যে তুমি কোডটি বুঝেছ, বিশেষ করে সেই ক্ষেত্রে যখন এটি জাভার মেথড কে ইনভোক করে। সেসকল মেথডের জন্য ডকুমেন্টশন পড়, এবং সাধারণ পরীক্ষামূলক কেসে পরীক্ষা করে দেখ। তুমি যা ভাবছ সেগুলো হয়ত ঘটবে না।

প্রোগ্রাম করার জন্য, প্রোগ্রামটি কি করবে তার জন্য তোমার একটি মানসিক মডেল প্রয়োজন। যদি এটি তোমার আশাতীত কাজ না করে, তাহলে সমস্যাটি প্রোগ্রামে নয়; এটা সম্ভবত তোমার মাথায়।

তোমার মানসিক মডেল সঠিক করার উত্তম উপায় হচ্ছে প্রোগ্রামটিকে ভিন্ন ভিন্ন অংশে ভাঙ্গা (সাধারণত ক্লাস এবং মেথড আলাদা করে ভাঙ্গা) এবং স্বাধীনভাবে পরীক্ষা করা। যখন তোমার মডেল এবং বাস্তবতার মাঝে অসামঞ্জস্যতা খুঁজে পাবে তখন তুমি এর সমাধান করতে পারবে।

এখানে কিছু সাধারণ লজিক ক্রুটি দেয়া হল:

- মনে রাখবে ইন্টিজার ডিভিশন সবসময় নিচের দিকে আবর্তন করে। যদি তুমি ভগ্নাংশ চাও, ডাবল ব্যবহার কর।

- ফ্লোটিং পয়েন্ট নাম্বার শুধুমাত্র আনুমানিক, সুতরাং নিখুঁত সঠিকতার উপর ভরসা করো না।

- আরও সাধারণভাবে, গণনাযোগ্য জিনিসের ক্ষেত্রে ইন্টিজার ব্যবহার কর এবং পরিমাপযোগ্য জিনিসের ক্ষেত্রে ফ্লোটিং পয়েন্ট নাম্বার ব্যবহার কর।

- তুমি যদি equality (==) অপারেটরের পরিবর্তে assignment (=) অপারেটর if, while এবং for শর্তে ব্যবহার করে থাক, তুমি সম্ভবত একটি এক্সপ্রেশন পাবে যা syntactically বৈধ কিন্তু শব্দার্থগত ভাবে ভুল।

- যখন তুমি equity (==) অপারেটর কোন অবজেক্টে ব্যবহার করবে, এটা আইডেন্টিটি পরীক্ষা করে। তুমি

যদি সমতা পরীক্ষা করতে চাও, তোমার equals মেথড ব্যবহার করা উচিত।

- ব্যবহারকারীর জন্য ডিফাইন করা, equals আইডেন্টিটি পরীক্ষা করে। তুমি যদি অন্য রকম সমতা চাও, এটিকে override করতে হবে।

- Inheritance সূক্ষ্ম লজিক ত্রুটি উৎপন্ন করতে পারে, কারন তুমি না বুঝেই inherited কোড রান করতে পার। উপরে "Flow of Execution" দেখ।

আমার একটি বৃহৎ এবং অমার্জিত এক্সপ্রেশন রয়েছে এবং আমি যা চাচ্ছি এটি তা করছে না।

জটিল এক্সপ্রেশন লিখা ভাল যতক্ষন পর্যন্ত না এটি পড়া সম্ভব, কিন্তু এটি ডিবাগ করা কঠিন হতে পারে। ভাল হয় যদি একটি জটিল এক্সপ্রেশনকে ভেঙে অ্যাসাইনমেন্টের সিরিজ বানিয়ে অস্থায়ী ভ্যারিয়েবল তৈরি করা যায়।

উদাহরনস্বরূপ:

```
rect.setLocation(rect.getLocation().translate(
-rect.getWidth(), -rect.getHeight()));
Can be rewritten as
int dx = -rect.getWidth();
int dy = -rect.getHeight();
Point location = rect.getLocation();
Point newLocation = location.translate(dx, dy);
rect.setLocation(newLocation);
```

ভাঙা সংস্করণটি পড়ার জন্য সহজ, কারন ভ্যারিয়েবলের নাম অতিরিক্ত ডকুমেন্টেশন প্রদান করে, এবং ডিবাগ করার জন্য সহজ, কারন তুমি অস্থায়ী ভ্যারিয়েবলের ধরন পরীক্ষা করতে পারবে এবং সেই সাথে তাদের মান প্রদর্শন করতে পারবে।

অন্য একটি সমস্যা যা বড় এক্সপ্রেশনের ক্ষেত্রে ঘটে তা হল যাচাই করার ক্রম তুমি যেরকম ভাববে সেরকম হবে না। উদাহরনস্বরূপ, $x/2\pi$ এর মান যাচাই করতে তুমি হয়ত লিখবে

```
double y = x / 2 * Math.PI;
```

এটি সঠিক নয়, কারন গুণ এবং ভাগ এর প্রাধাণ্যতা একই সাথে, এবং তাদের যাচাই করা হবে বাম থেকে ডানে। এই এক্সপ্রেশনটি গণনা করবে $x\pi/2$.

যদি তুমি অপারেটরের অপারেশন সম্পর্কে নিশ্চিত না হও তাহলে বন্ধণীর ব্যবহার কর।

```
double y = x / (2 * Math.PI);
```

এই সংস্করণটি সঠিক, এবং অন্যান্য লোকের পড়ার জন্য সহজ যারা অপারেটরের প্রাধাণ্যতা মনে রাখতে পারে না।

আমি যা চাই আমার মেথড তা রিটার্ন করে না।

যদি তোমার জটিল এক্সপ্রেশনের সাথে একটি রিটার্ন স্টেটমেন্ট থাকে, তাহলে রিটার্ন করার পূর্বে তুমি ভ্যালু প্রিন্ট করার সুযোগ পাবে না। আবারও বলছি, তুমি একটি অস্থায়ী ভ্যারিয়েবল ব্যবহার করতে পার। উদাহরনস্বরূপ, নিচের কোডের পরিবর্তে:

```
public Rectangle intersection(Rectangle a, Rectangle b) {
        return new Rectangle(
                Math.min(a.x, b.x),
                Math.min(a.y, b.y),
                Math.max(a.x+a.width, b.x+b.width)-Math.min(a.x, b.x)
                Math.max(a.y+a.height, b.y+b.height)-Math.min(a.y, b.y) );
}
```

তুমি লিখতে পার

```
public Rectangle intersection(Rectangle a, Rectangle b) {
        int x1 = Math.min(a.x, b.x);
        int y2 = Math.min(a.y, b.y);
        int x2 = Math.max(a.x+a.width, b.x+b.width);
        int y2 = Math.max(a.y+a.height, b.y+b.height);
        Rectangle rect = new Rectangle(x1, y1, x2-x1, y2-y1);
        return rect;
}
```

এখন তোমার সুযোগ রয়েছে রিটার্ন করার পূর্বে যে কোন intermediate ভ্যারিয়েবল প্রদর্শন করার। এবং x1 ও y1 পুনরায় ব্যবহারের মাধ্যমে তুমি কোডকে ছোটও করে ফেলেছ।

আমার প্রিন্ট স্টেটমেন্ট কিছুই করছে না

যদি তুমি println মেথড ব্যবহার করে থাক, তাহলে তৎক্ষণাত আউটপুট প্রদর্শিত হবে, কিন্তু যদি তুমি print ব্যবহার কর (অন্তত কিছু পরিবেশে) নতুন newline প্রদর্শিত হওয়ার আগ পর্যন্ত আউটপুট জমা হবে কিন্তু প্রদর্শিত হবে না। যদি প্রোগ্রামটি নতুন লাইন প্রদর্শন না করেই বন্ধ হয়ে যায়, তুমি হয়ত জমা করা আউটপুট আর দেখতে পাবে না।

যদি তুমি সন্দেহ কর এটা ঘটছে তাহলে, সবগুলো অথবা কিছু print স্টেটমেন্ট println এ পরিবর্তন করে আন।

আমি আটকে গিয়েছি, আমার সাহায্য দরকার

প্রথমত, কম্পিউটার থেকে কিছু সময়ের জন্য দূরে যাও। কম্পিউটার কিছু তরঙ্গ নিঃস্বরণ করে যা ব্রেইনকে প্রভাবিত

করে, এবং নিচের কিছু লক্ষণ দেখা যায়:

- নৈরাশ্য এবং ক্রোধ।
- কুসংস্কারাচ্ছন্ন চিন্তা ভাবনা (যেমন: কম্পিউটার আমাকে ঘৃণা করে) এবং যাদুকরী চিন্তা ভাবনা (যেমন: যদি আমি নীল রঙের টুপি পড়ে প্রোগ্রাম করি তাহলেই শুধুমাত্র প্রোগ্রাম কাজ করে)।
- আঙ্গুরফল টক ("এই প্রোগ্রামটি এমনিতেই ভাল ছিল না")।

যদি তোমার মধ্যে উপরের যে কোন লক্ষণ দেখা যায়, উঠে যাও এবং হাটাচলা করে এসো। যখন তুমি শান্ত হবে, প্রোগ্রাম নিয়ে ভাব। এটা কি করছে, এরকম করার সাম্ভাব্য কারন কি? কাজ করেছে এমন প্রোগ্রাম সর্বশেষ কখন দেখেছিলাম? এবং তারপর আমি কি করেছিলাম?

মাঝে মাঝে বাগ খুঁজে পেতে একটি সময় লাগে। আমি প্রায়শই যখন হাটাচলা করি তখন চিন্তা করে পাই বাগ কোথায় রয়েছে। বাগ খুঁজে পাওয়ার উত্তম জায়গা হচ্ছে বাস, শাওয়ার এবং বিছানা।

না, আমার সত্যিই সাহায্য প্রয়োজন।

এটা ঘটে। এমনকি বড় বড় প্রোগ্রামাররাও আটকে যায়। মাঝে মাঝে তোমার দরকার হবে সজীব দুটি চোখ। কাউকে আনার পূর্বে নিশ্চিত হয়ে নাও উপরের সকল টেকনিক তুমি খাটিয়েছ। তোমার প্রোগ্রামটি যতটুকি সম্ভব ছোট হতে হবে, এবং তুমি কাজ করতে থাকবে ছোট একটি ইনপুট নিয়ে যার জন্য ক্রুটি ঘটছে। তুমি নিশ্চই প্রিন্ট স্টেটমেন্ট জায়গামত লিখেছ (এবং যে আউটপুট আসছে তা বোধগম্য)। তুমি অবশ্যই ভালভাবে প্রোগ্রামটি বুঝবে যাতে পরিষ্কার ভাবে অন্যকে বুঝাতে সক্ষম হও।

যখন তুমি কাউকে সাহায্য করার জন্য আনবে, তার যেই তথ্য প্রয়োজন সেটা দাও।

- এটা কি ধরনের বাগ? সিনট্যাক্স, রান-টাইম, অথবা লজিক?
- ক্রুটি ঘটার পূর্বে সর্বশেষ কোন কাজটি করেছ? সর্বশেষ কোডের কোন লাইনটি লিখেছ? অথবা নতুন পরীক্ষামূলক কেস কোনটি? যেটা ব্যর্থ হয়েছে?
- যদি বাগটি কম্পাইল-টাইম অথবা রান-টাইমে ঘটে থাকে, ক্রুটির বার্তাটি কি, এবং কোডের কোন অংশকে এটা নির্দেশিত করে?
- তুমি কি কি চেষ্টা করেছ? এবং কি শিখেছ?

তুমি যখন তাকে সমস্যা ব্যাখ্যা করতে থাকবে, তখন তুমি নিজেই হয়ত সমাধান পেয়ে যেতে পার। এই বিষয়টি এতই সাধারণ যে কিছু মানুষ এটির জন্য একটি ডিবাগিং টেকনিক বের করেছে এবং নাম দিয়েছে "rubber ducking"। এটা কিভাবে কাজ করে দেখ:

১। একটি রাবার ডাক (rubber duck) ক্রয় কর।

২। যখন তুমি সত্যি সত্যিই কোন সমস্যায় পড়বে, রাবারের হাঁস টিকে টেবিল উপর রাখ এবং বল, "রাবারের হাঁস, আমি এটি সমস্যায় আটকে গিয়েছি। যা ঘটে তা এরকম..."

৩। রাবারের হাঁস কে সমস্যা ব্যাখ্যা কর।

৪। এবং সমাধান খুঁজে নাও।

৫। রাবারের হাঁস কে ধন্যবাদ দাও।

আমি কিন্তু মজা করছি না। এটা দেখ: http://en.wikipedia.org/wiki/Rubber_duck_debugging

আমি বাগ পেয়েছি!

যখন তুমি বাগ পেয়ে যাবে, এটা নিশ্চই পরিষ্কার যে কিভাবে এটা ঠিক করতে হবে। কিন্তু সবসময় নয়। মাঝে মাঝে যেটা আসলে বাগ সেটা শুধু ইঙ্গিত করে তুমি প্রোগ্রামটি বুঝনি, অথবা তোমার অ্যালগরিদমে ত্রুটি রয়েছে। এই ক্ষেত্রে, তোমকে অ্যালগরিদম নিয়ে পুনরায় চিন্তা করতে হবে, অথবা তোমার মানসিক কাঠামোটি ঠিকঠাক করতে হবে। কম্পিউটার থেকে দূরে গিয়ে সময় নিয়ে কিছুক্ষণ ভাব, টেস্ট কেস গুলো খাতা কলম নিয়ে কর অথবা কম্পিউটেশন উপস্থাপনার জন্য ডায়াগ্রাম আঁক।

বাগ ঠিক করার পর, নতুন ত্রুটি বানানো শুরু করে দিও না। কিছু সময় নাও এবং ভাব এটা কি ধরনের বাগ ছিল, কেন তুমি ত্রুটি ঘটিয়েছিলে, কিভাবে ত্রুটি মুক্ত হয়েছে এবং এটা আরও দ্রুত খুঁজে পেতে তুমি কি করতে পারতে। পরবর্তী সময়ে তুমি একই রকম কিছু দেখলে দ্রুত বাগ খুঁজে পাবে এবং সমাধান করতে পারবে।

ইনডেক্স